AD-A033 437 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
MULTICS OLPARS OPERATING SYSTEM.(U)
SEP 76 D B CONNELL, K N KLINGBAIL F30602-75-C-0226
UNCLASSIFIED PAR-74-25-B RADC-TR-76-271-VOL-2 NL

1 OF 7
AD
A033437

RADC-TR-76-271, Vol II (of two)
Final Technical Report
September 1976

MULTICS OLPARS OPERATING SYSTEM

Pattern Analysis and Recognition Corporation

Approved for public release;
distribution unlimited.

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

add A+B after 74-25 vol 1+2 per Originator, (315) 336-8400 M Crumbacker 14 Jan 77

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public including foreign nations.

This report has been reviewed and is approved for publication.

APPROVED: Patricia J. Baskinger
PATRICIA J. BASKINGER
Project Engineer

APPROVED: Robert D. Krutz
ROBERT D. KRUTZ, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss
JOHN P. HUSS
Acting Chief, Plans Office

Readers of this report are advised that some of the illustrations included herein are relatively poor quality reproductions of computer printouts. They are, however, the best available.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

19 18

1. REPORT NUMBER: RADC-TR-76-271-Vol II (of two) - 2

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

9

4. TITLE (and Subtitle): MULTICS OLPARS OPERATING SYSTEM.

6

5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. Jun[redacted]73 - Jun[redacted]76

6. PERFORMING ORG. REPORT NUMBER: 74-25-B

7. AUTHOR(s): David B. Connell, Richard A. Jackson, Kermit N. Klingbail, et al

10 15

8. CONTRACT OR GRANT NUMBER(s): F30602-75-C-0226, F30602-73-C-0352

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Pattern Analysis and Recognition Corporation, 228 West Dominick St, Rome NY 13440

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 55971309

11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISCP), Griffiss AFB NY 13441

11

12. REPORT DATE: Sep~~tember~~ 1976

13. NUMBER OF PAGES: 658

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same

12 660p.

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report): Approved for public release; distribution unlimited.

16 5597

14 PAR-74-25-B

17 13

Per Tel Telecon Recd from PAR 14 Jun 79

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report): Same

18. SUPPLEMENTARY NOTES: RADC Project Engineer: Patricia J. Baskinger

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):

Pattern Recognition
Pattern Analysis
Decision Theory
Classification
Clustering
Measurement Evaluation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The development of interactive graphics computer systems for use in the detection, identification, and transformation of patterns contained in high-dimensional data has been a continuing program at the Rome Air Development Center since 1968 (RADC-TR-70-139; RADC-TR-71-177; RADC-TR-73-241). This long standing effort has resulted in the implementation of OLPARS (the On-Line Pattern Analysis and Recognition System), IFES (the Image Feature Extraction System), and WPS (the Waveform Processing System). This report contains detailed design and user-oriented information related to MOOS (the MULTICS OLPARS Operating

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

390 101

LB

System), an advanced version of OLPARS currently resident upon the Honeywell 6180 MULTICS computer system. The currently operational system represents an implemented version of the operations described in a previous report (RADC-TR-73-241); appropriate selections of that report are retained within this document. This report contains brief descriptions of the MOOS system and the mathematics underlying the system algorithms. A major portion of this document is reserved for a user's manual (providing detailed information relating to the operation of all system options) and for MOOS program documentation.

PREFACE

This is Volume II of a two-volume Final Report by Pattern Analysis and Recognition Corporation, 228 W. Dominick Street, Rome New York and represents work performed under Contracts F30602-75-C-0226, F30602-73-C-0352, and F30602-73-C-0351, Job Order Numbers 55971309, 55971306, and 55971305 for the Rome Air Development Center, Griffiss Air Force Base, New York. Mr. Jonn C. Faust and Mrs. Patricia J. Baskinger were the RADC Project Engineers.

TABLE OF CONTENTS

VOLUME II

SECTION 3

MOOS FILING SYSTEM

3.1. INTRODUCTORY REMARKS

The MOOS Filing System has been developed to allow multiple user operations using the OLPARS subset under MULTICS. Each user is provided with a temporary data storage area as well as a set of more permanent data files. The temporary area (the process directory) contains his current system description (files "sysdata", "scratch" and "display") and his current data trees. His permanently assigned area (the working directory) provides file entries for data which may be utilized on a day-to-day basis as well as a hardcopy dump area for delayed print-out. In addition to the permanent user area, the central system contains the object programs available under OLPARS and a data storage area from which data may be transferred into any user's temporary data area. Under the MULTICS structure, each user has access to the programs in the central system directory (the default working directory for all users) for operations upon data in his own process directory. Source programs for MOOS are stored in the central system directory. System programmers may add and modify programs from "OLPARS" to produce object versions within that directory.

Within the user's process directory each data tree is represented by a file under the name of the data tree and a set of data files named via a concatenation of a data tree character and a data class name. The data tree characters are represented by upper case letters from A through T; a system limit of 20 trees is imposed.

A variety of structured data files have been designed to accommodate data in the OLPARS format. These files are created and manipulated in the user's process directory by the system programs and their existence is transparent to the system user. Therefore, multiple users may operate the MOOS programs on independent data sets. The files will be described in the following order:

Standard Process Directory Files (these files exist when the user is executing MOOS programs and are deleted at signoff time).

- o "sysdata" file (Section 3.2.1.)
- o "scratch" file (Section 3.2.2.)
- o "display" file (Section 3.2.3.)
  - a) one-space display
  - b) two-space display
  - c) rank order display
  - d) confusion matrix display

Variable Process Directory Files (each user has a number of these files in existence at any time MOOS programs are being executed; they are deleted at signoff time).

- o TREENAME files (Section 3.3.1)
- o DATACLASS files (Section 3.3.2.)
- o MOOSLOGIC files (Section 3.3.3.)

  a) header and structure area
  b) pairwise logic file
  c) grouped partition logic - 1-space
  d) grouped partition logic - 2-space
  e) grouped Boolean logic
  f) nearest mean vector logic
  g) closed decision boundary logic
  h) independent Boolean reject logic
  i) logic error files

Standard Central System Directory Files and Directories (each of these files is maintained permanently in the central system directory).

- o "option_file" file (Section 3.4.1)
- o "trandata" directory (Section 3.4.2)
- o "seg_o_trees" file (Section 3.4.2)
- o "olpars" archive

Standard User Working Directory Files and Directories (each user is allowed to store data in a permanent directory; data from this directory are deleted only by user command).

- o "saved_trees" directory (Section 3.5.1)
- o "seg_o_trees" file (Section 3.5.2)
- o "vectors" file (Section 3.5.3)
- o "seg_o_logic" file (Section 3.5.4)

3.2. STANDARD TEMPORARY DIRECTORY FILES

3.2.1. "sysdata" File

The "sysdata" file (Figure 3-1) is created upon initialization of the MOOS System and serves as the primary system data index for the user. Major information headings to be found in "sysdata" are: (1) current system status (64 words); (2) a list of data trees currently maintained by the system (80 words); and (3) a tree structure table indexing all data classes currently being maintained by the system (3948 words). The system status (CSS) section is maintained in the first 64 words of the file. Each system program is responsible for maintaining accurate current entries in the CSS block. The section containing the list of data trees (FOREST) will be updated by any system program responsible for data input, transformation or deletion. A maximum of twenty (20) trees may be maintained concurrently. Finally, the portion containing the tree structure tables (SCHOOL) is an open-ended section which will link each data class in the system to its appropriate tree and provide structure to each tree. Virtually all system functions will require information from this block and therefore a number of subroutines which facilitate access to this portion of "sysdata" have been described below.

The following parameters are contained in the CSS block as depicted graphically in Figure 3-1.

CSS1 The data tree currently being analyzed. An eight-character string using two computer words.

CSS2 The data class name currently being analyzed. A four-character string using one computer word (the string "****" signifies the senior node of the current tree).

CSS3 The current tree index number. An integer value representing the current data tree. The legal value ranges from 1 to 20. The tree character may be found by adding 100_8 to the tree index representing the tree, thus converting the tree index to an ASCII character between A and T (101_8 to 124_8).

CSS4 The last node entry index. A fixed binary (integer) number referring to the highest slot number which contains meaningful information in the SCHOOL.

CSS5 The system option currently operating (this item must be reset by each major system program). A fixed binary (integer) number indexing a location in the "option-file" list (see Section 2, subroutine "option").

	Word				
CSS	1	CSS1			Current Tree Name
	2	CSS1			
	3	CSS2			Current Class Name
	4	CSS3	CSS4	CSS5	Tree/Last Node/Option
	5	CSS6			Sense Switches
	6	CSS7			Temporary Buffer Area
	62	CSS8			2-Space Cutoff Value
	63	CSS9			1-Space Bin Factor
FOREST	64				
	65	F1			Tree Name
	66	F1			
	67	F2	F3	F4	ndim/numclass/firstclass
	68	F5			No. Vectors
	144	Repeated Tree Entries			
SCHOOL	145	S1			Class Name
	146	S2	S3	S4	ndim/numclass/firstclass
	147	S5			No. Vectors
	148	S6	S7	S8	depth/seniorclass/ nextclass
	n	Repeated Class Entries			

Figure 3-1: "sysdata" File Format

CSS6 36 Sense Switches which may be utilized to direct variable system operations under the "sense n v" option, where n is an integer value from 1 to 36 and designates the sense switch to be set, and v represents the value (0 or 1) to which it will be set. See the user's manual write-up on sense for a description of the various uses of the sense switches.

CSS7 A temporary buffer area.

CSS8 The cutoff value for the two-space display stored in integer format. A two-space cluster display will be generated automatically if the number of vectors to be displayed is greater than the value of CSS8 (otherwise a two-space scatter display will be generated).

CSS9 The bin factor for the one-space display stored in integer format. The bin factor determines the number of columns in the one-space display.

The parameters listed below describe one entry in the FOREST block. Twenty (20) null tree entries are defined in this block by the system initialization routine.

F1 The TREENAME associated with this FOREST entry. An eight-character string using two computer words. Null entries (entries for which no tree currently exists within the system) are denoted by the character string "notatree" in this parameter.

F2 The number of dimensions for the data tree in this entry. There is no requirement in MOOS to maintain a consistent number of measurements among trees currently active within the system. The twelve (12) high-order bits of word 3 of each tree entry are associated with this parameter and the format is fixed binary (integer).

F3 The number of nodes in the next lowest tree level is contained in the central 12 bits of word 3 of each tree entry in fixed binary (integer) format. By definition, this item is the number of a priori data classes input with the data tree.

F4 The index of the first node entry associated with the data tree. A fixed binary (integer) value in the lowest twelve (12) bits of each data tree entry.

F5 The number of vectors in the entire data tree. A fixed binary (integer) number occupying one full computer word.

The parameters for one entry in the SCHOOL block are described below. All data classes for all trees in the system are defined within this block, linked together via pointers within each entry. The CSS4 parameter indicates the current extent of this table.

S1 The CLASSNAME associated with this SCHOOL entry. A four-character string using one computer word. Null entries (entries for which data classes once existed but have been deleted) are denoted by the character string "nono" in this parameter.

S2 The number of dimensions for the data class in this entry. The twelve (12) high-order bits of word 2 of each node entry are associated with this parameter; the format is fixed binary (integer).

S3 The number of data classes in the next lowest tree level is contained in the central twelve (12) bits of word 2 of each class entry in fixed binary (integer) format. This parameter will be set by the structure analysis partition routine during data class partition operations.

S4 The <u>index</u> of the first node associated with the subsets of this data class. A fixed binary (integer) value in the lowest twelve (12) bits of each data tree entry.

S5 The number of data vectors contained in this data class. A fixed binary (integer) value occupying one full computer word.

S6 The level (or depth) of the data class within the data tree hierarchy. The senior node is assigned depth 1, the a priori nodes depth 2, etc. This value is contained in the twelve (12) higher order bits of word 4 within each entry.

S7 The <u>index</u> of the senior data class associated with this data set. For each a priori data class, then, this value points to the entry in the FOREST block associated with the appropriate data tree.

S8 The <u>index</u> of the next class on the current level which is associated with this tree. The final class within each level will contain a value of 0 in parameter S8.

3.2.2. <u>"scratch" File</u>

The "scratch" file is maintained in each user's temporary storage area. "scratch" is utilized to pass transient data from one internal system module to another and to provide variable length arrays as necessary to prevent unneeded system limitations.

Since "scratch" is maintained in a virtual memory environment, array assignments within this file will be generated by no other limitations than program and data requirements. Although no specific "scratch" arrangement may be expected upon entrance to a user-specified program, internal system programs may require a detailed setting of as much of the scratch array as is needed to perform its computations.

3.2.3. "display" File

The "display" file contains all information necessary to regenerate the last system display produced. Four major types of system displays have been defined: one-space data plots (with microview and macroview versions), two-space data plots (scatter and cluster plots), rank order measurement displays (as in the measurement evaluation operation under 1604B OLPARS), and confusion matrix displays for logic evaluation operations. Each program wishing to modify the "display" file may determine the appropriateness of its action by checking word 1 of the file (system display code: 0 - no display; 1 - two-space display; 2 - rank order display; 3 - confusion matrix; 4 - one-space display). When a display program is called, word 1 may be checked to determine whether it is to generate a new display (word 1 = 0) or regenerate an old one (word 1 = n , where n is the display number assigned to the routine).

Two-space display (display code = 1) settings: The system two-space displays (cluster and scatter plots) utilize two files ("display" and "csdata") to generate the data mapping. The "csdata" file is a list of x and y data projections. (Note: these two files are utilized in the same manner for the one-space data projections but their use is transposed - "display" is the data projection file).

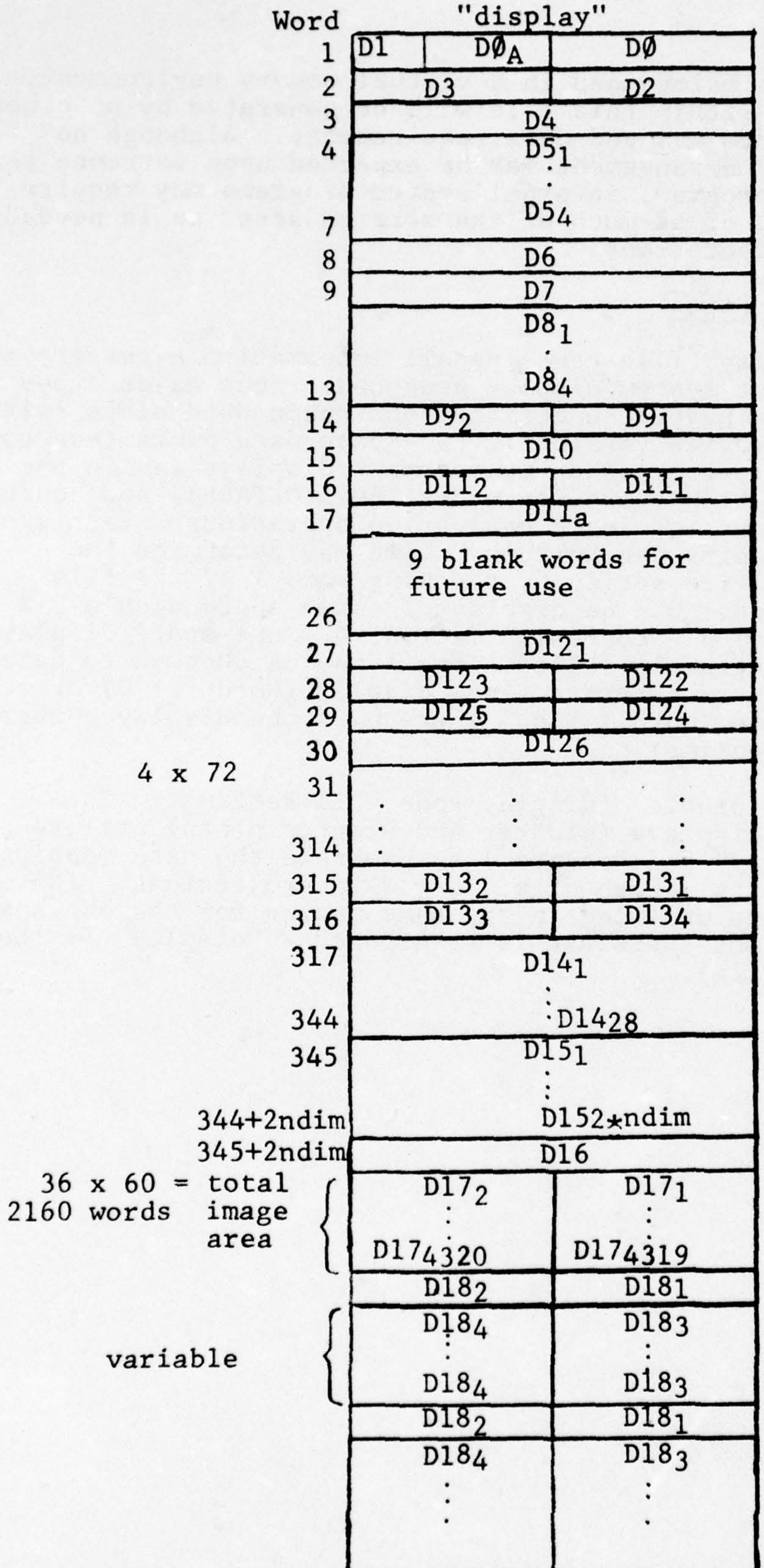


Figure 3-2: <u>two space "display" file format</u>

DØ	system display code
	Ø not-set 1 2-space 2 rank-order 3 confusion matrix 4 1-space
$DØ_A$	temporary symbol
D1	tree character
D2	dimensionality
D3	number of classes
D4	data set projection flag
	0 - data set needs to be projected and min and max calculated
	1 - data set needs to be projected and min and max supplied
	2 - data set already projected
$D5_1$-$D5_4$	"original" min's and max's stored as xmin, xmax, ymin, ymax
D6	cluster/scatter flag
	0 cluster plot 1 scatter plot
D7	relative position flag
	0 - data needs to be positioned on 60 x 36 grid
	1 - data already positioned
$D8_1$-$D8_4$	"current" min's and max's
$D9_1$	sequence flag
	0 - not set
	1 - set
$D9_2$	sequence number - total number of times a given display has been sequenced.

D10 cycle option

0 - not set

1 - block plot 2-skip plot

D11 (if D10 ≠ 0)

if D10 = 1	$D11_1$ = block size	$D11_2$ = 0
if D10 = 2	$D11_1$ = starting no.	$D11_2$ = skip size

$D11_a$ option call flag

0 - call option

1 - return

$D12_1$ classname

$D12_2$ eliminate

0 - no

1 - yes

$D12_3$ intensify

0 - no

1 - yes

$D12_4$ csdata index; index to projected data in csdata for this class

$D12_5$ relpos index, index to relative position data in display for this class

$D12_6$ number of vectors this class

$D13_1$ number of boundaries 0,1,2

$D13_2$ 0 - not set

1 - set, call redraw to draw boundaries

$D13_3$ number of boundary points in boundary #1

$D13_4$ number of boundary points in boundary #2

$D14_1$-$D14_{28}$ x,y coordinates of boundary #1 and convex pt. #1

x,y coordinates of boundary #2 and convex pt. #2

$D15_1$,-$D15_{2*ndim}$ x,y projection vectors

D16 number of points in projection image

D17 cluster image 2160 words

$D17_1$ class symbol $D17_2$ intensify 0 - no 1 - yes

D18 relative position (relpos) area

$D18_1$ total number of entries for each class

$D18_2$ number of distinct relative positions for each class

$D18_3$ relative position

$D18_4$ number of vectors at this relative position

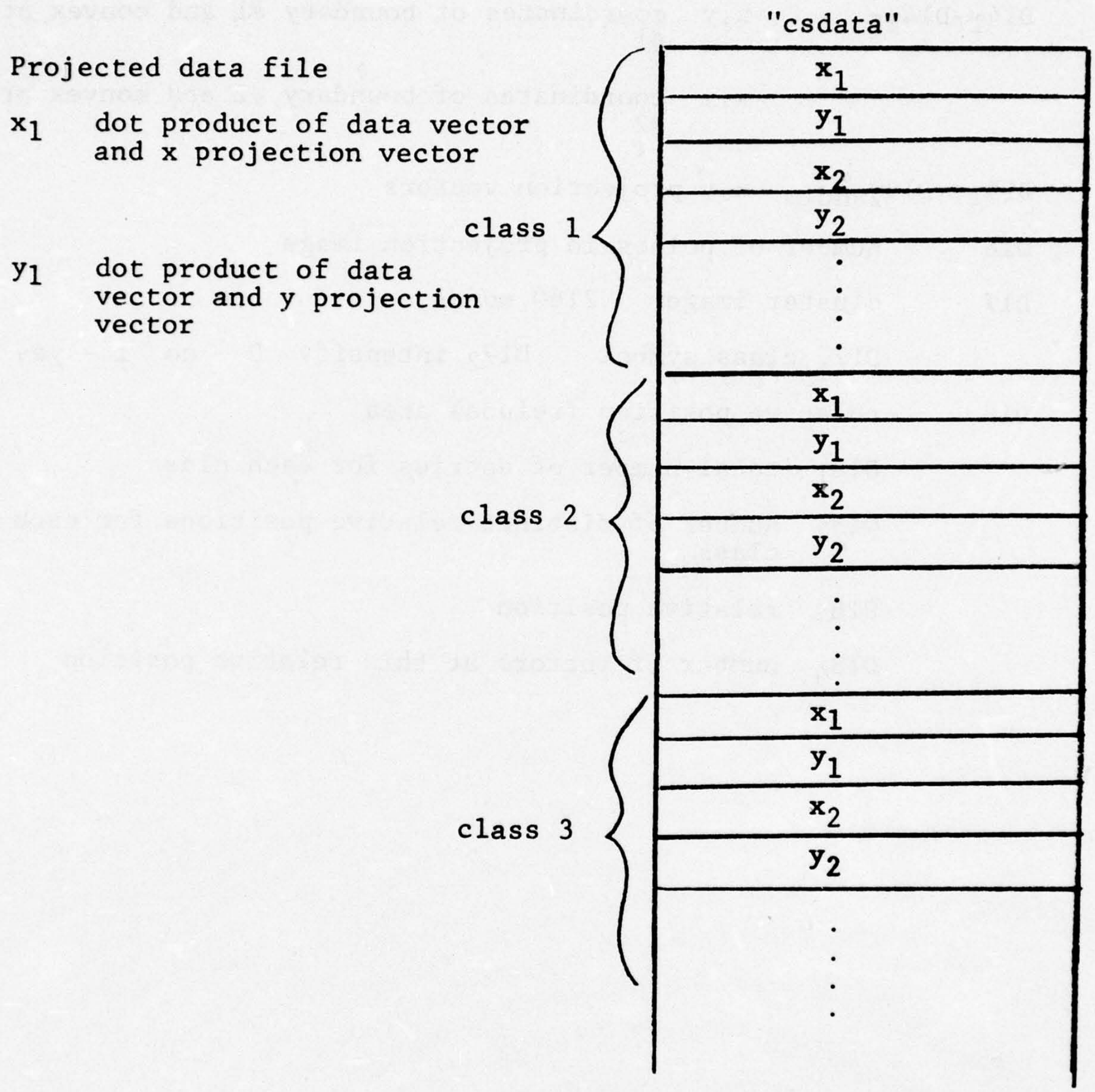


FIGURE 3-3: TWO SPACE "csdata" FILE FORMAT

Word	Rank Order	
1	$D1_a$ | $D1_b$	$D1_b$=system display code
2	D2	ndim
3	D3	sort order
4	$D4_a$ | $D4_b$	$D4_a$=tree character
5	D5	ncls
6	$D6_1$	data class names
	⋮	
5+ncls	D6 ncls	
6+ncls	$D7_1$ | $D8_1$ | $D9_1$ | $D10_1$	display information (see description)
	⋮	
5+ncls+ndim	$D7_{ndim}$ | $D8_{ndim}$ | $D9_{ndim}$ | $D10_{ndim}$	
6+ncls+ndim	$D11_1$	display sorting area
	$D12_1$	
	⋮	
	$D11_{ndim}$	
5+ncls+3*ndim	$D12_{ndim}$	
6+ncls+3*ndim	$D13_1$	$D(X_p)$
	⋮	
5+ncls+4*ndim	$D13_{ndim}$	
6+ncls+4*ndim	$D14_{1,1}$	$D_{class(i)}(X_p)$
	⋮	
	$D14_{ndim,1}$	
	⋮	
5+ncls+(4+ncls)*ndim	$D14_{1,ncls}$	
6+ncls+(4+ncls)*ndim	⋮	
	$D14_{ndim,ncls}$	
	$D15_{1,2,1}$	$D_{class(i),class(j)}(X_p)$
	⋮	
	$D15_{ndim,2,1}$	
	⋮	
	$D15_{1,ncls,1}$	
	⋮	
$\left\{5+ncls+\left(4+ncls+\frac{ncls*(ncls-1)}{2}\right)*ndim\right\}$	$D15_{ndim,ncls,ncls-1}$	

Figure 3-4: Rank Order "display" file format

o <u>Rank Order Display (display code = 2) Settings:</u>

$D1_a$ (used by hardcopy routine) 18 bits

= 0 if current option is rnk$oall
= 1 if current option is rnk$mbc or rnk$mbcp
= 2 if current option is rnk$bcls
= 3 if current option is rnk$bycp

$D1_b$ system display code 18 bits

0 - if D7, D8, D9, D10 are to be recomputed
2 - Rank-order display

D2 dimensionality of data

D3 order of sorting information

0 - ascending
1 - descending

$D4_a$ tree character

$D4_b$ (used by hardcopy routine) 27 bits

= measurement number if $D1_a = 1$
= class character if $D1_a = 2$
= classpair (char/char) if $D1_a = 3$

D5 number of classes

D6 data class names to take place in display

D7-D10 information to be displayed on screen:

D7 "*" if "*" has been turned on
D8 class best destinguished by this measurement
D9 & D10 classpair best distinguished by this measurement

D11&D12 area in which data are sorted for display

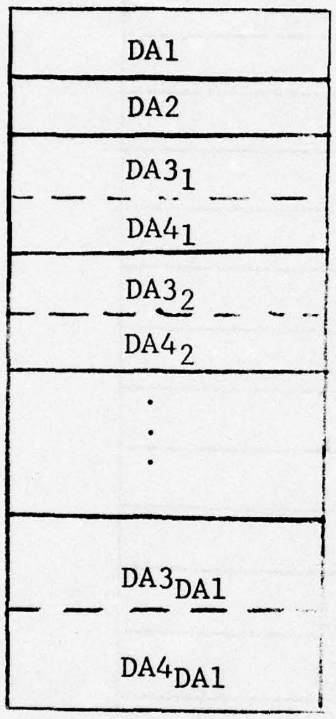
D11 relative index into D7-D10 block to show to which measurement the associated D12 value corresponds
D12 value to be sorted

D13 measurement discrimination for measurement X_p

D14 measurement discrimination for class(i) on measurement X_p

D15 measurement discrimination for classpair i,j on measurement X_p

"Displaya" is a file created by "rnk$mbc" or "rnk$mbcp" which contains information for ranking of measurements. It is, in effect, an extension of the rank order display file.



DA1 — Number of entries in file

DA2 — The number of the last printed entry

$DA3_1$ - classname or classpair name of entry i

$DA4_1$ - value associated with current number of class or classpair $DA3_i$

Figure 3-5: Rank Order "display" File Format

Confusion Matrix "display" file format

1	C1
2	C2
3	C3
4	C4
5	C5
6	C6
7	C7
8	C8
9	C9
10	C10
11	C11
12	C12
13	C13
14	C14
15	C15
17	C16
18	C17
19	$C18_1$ . . .
18 + tncls	$C18_{tncls}$

Confusion Matrix "display" file format (continued)

91	$C19_1$
92	$C20_1$
93	$C21_1$
94	$C22_1$
95	$C23_{1,1}$ . . $C23_{1,\ lncls}$
94 + lncls	$C24_1$
95 + lncls	$C25_1$
97 + lncls	$C26_1$
	. . . .
	$C19_{tncls}$
	$C20_{tncls}$
	$C21_{tncls}$
	$C22_{tncls}$
	$C23_{tncls,1}$ . . $C23_{tncls,\ lncls}$
	$C24_{tncls}$
	$C25_{tncls}$
90 + tncls * (lncls + 7)	$C26_{tncls}$

Confusion Matrix "display" file format (continued)

91 + tncls * (lncls+7)	C27
116 + tncls * (lncls+7)	$C28_1$ . . . $C28_{lncls}$
116 + tncls * (ncls+7) + lncls	$C29_1$ . . . $C29_{tncls}$
116 + tncls * (lncls+8) + lncls	C30
	$C31_1$
	$C32_1$
	. . .
	$C31_{C30}$
	$C32_{C30}$
117 + tncls * (lncls+8) + lncls+ (2*C30)	$C33_1$ . . . $C33_{128}$

Figure 3-6: Confusion Matrix "display" file format

o Confusion Matrix display (display code=3) settings:

C1 - system display code (3 for confusion matrix)
C2 - an integer representing the display type.
  0 indicates confusion matrix summary
  1 indicates confusion matrix to the screen
  2 indicates confusion matrix to the printer
  4 indicates confusion matrix summary to go to the screen only
C3 - number of true classes (tncls)
C4 - number of dimensions
C5 - not used
C6 - total number of vectors
C7 - total number correct
C8 - total number errors
C9 - total number rejects
C10 - overall % correct
C11 - overall % error
C12 - overall % reject
C13 - number of assigned classes (lncls)
C14 - set to 1 for overall evaluation, 2 for partial pairwise evaluation, 3 for partial nearest mean vector evaluation, 5 for partial closed decision boundary evaluation, and 6 for subroutine logic evaluation
C15 - 8-character treename of the data set on which logic was designed
C16 - 4-character nodename of the data set on which logic was designed
C17 - reject flag - set to 1 if there were rejects, 0 if no rejects
C18 - an array of length tncls containing the 4-character nodenames of the true classes
C19 - number of vectors in each true class
C20 - number of correctly classified vectors in each true class
C21 - number of errors in each true class
C22 - number of rejects in each true class
C23 - an array of length lncls containing a "column" of the confusion matrix
C24 - % correct for each true class
C25 - % error for each true class
C26 - % reject for each true class
C27 - an array of 25 unused words
C28 - an array of length lncls containing the 4-character nodenames of the assigned classes
C29 - an array of length tncls containing the number of favorably broken ties for each true class
C30 - number of pairwise logic nodes
C31 - a pairwise logic node number
C32 - the threshold vote count set for the preceding C31 entry

Note: if C30 is zero, then C31 and C32 entries are left out, i.e. the C33 array follows C30.

C33 - an array of length 128 containing the 4-character classnames which correspond to each logic node. This array allows logic node names to be indexed by logic node number.

o <u>One-Space Display (display code = 4) Settings</u>:

Word			
1	C2	C1	C0
2–9	C_a		
10–16	unused		
17	C_b		
18	C_c		
19	C_d		
20	C_e		
21	C_f		
22	C3		
23	C4		
24	C5		
25	C6		
26	C7		
27	C8		
28	$C9_1$		
29	$C9_2$		
30	$C10_1$		
31	$C10_2$		
32	$C11_1$		
33	$C11_2$		
34	unused		
35	C12		
36	$C13_1$		
37	$C13_3$	$C13_2$	
38	$C13_5$	$C13_4$	
39	$C13_6$		
40			

	⋮
323	⋮
324	$C14_1$
325	$C14_2$
326	$C14_3$
327	$C15_1$
	⋮
326+ndim	$C15_{ndim}$
327+ndim	C16
328+ndim	$C17_0$
329+ndim	$C17_1$
	⋮
	$C17_{nbin}$
329+ndim+nbin	$C17_0$
330+ndim+nbin	$C17_1$
	⋮
329+ndim+2*nbin	$C17_{nbin}$
330+ndim+2*nbin	⋮

Figure 3-8: <u>One-Space Display "csdata" format</u>

C4	number of classes
C5	view 0 - macro 1 - micro
C6	type of scaling 0 - probabilities 1 - counts
C7	data projection flag 0 - data needs to be projected and (min. and max.) calculated 1 - data needs to be projected and (min. and max.) supplied 2 - data already projected 3 - min. and max. supplied by calling routine (the range used is the range of the data or the supplied range, whichever is greater)
C8	sort flag 0 - data needs to be sorted into bins 1 - data already sorted
$C9_1$	"original" xmin
$C9_2$	"original" xmax
$C10_1$	"current" xmin
$C10_2$	"current" xmax
$C11_1$	sequence flag 0 - not set 1 - set
$C11_2$	sequence number (# of times sequenced)
C12	number of bins (nbin)
$C13_1$	classname
$C13_2$	eliminate 0 - no 1 - yes
$C13_3$	intensify 0 - no 1 - yes
$C13_4$	index to this class' binned data ($C17_0$)
$C13_5$	index to this class' projected data in "display"
$C13_6$	number of vectors of this class the C13 block repeated for a possible 72 classes

$C14_1$ number of boundaries

$C14_2$ x-coordinate of boundary 1

$C14_3$ x-coordinate of boundary 2

$C15_1$ - $C15_{ndim}$ projection vector

C16 max. prob/count of all displayed classes

C17 is "bin area" and repeated for each class

$C17_0$ - max prob of this class

$C17_1$ - prob of each bin

⋮

$C17_{nbin}$

M0	number of entries	
$M1_0$	class name	
$M1_1$	intensify	0 - no 1 - yes
$M1_2$	number of vectors in class in range	
$M1_3$-$M1_{2+nbinc}$	prob/count of each bin	
$M2_0$	class name	
$M2_1$	intensify	
$M2_2$	number of vectors in class in range	
$M2_3$-$M2_{2+nbinc}$	prob/count of each bin	

Word	
1	M0
2	$M1_0$
3	$M1_1$
4	$M1_2$
	$M1_3$
4+nbin	$M1_{2+nbin}$
5+nbin	$M2_0$
6+nbin	$M2_1$
7+nbin	$M2_2$
8+nbin	$M2_3$
	$M2_{2+nbin}$
8+2*nbin	$M3_0$
. . . .	. . . .

Figure 3-9: One-Space Micro View "microbuff" format

3.3. VARIABLE TEMPORARY DIRECTORY FILES

3.3.1. TREENAME Files

The TREENAME files are created during data input or transformation. They consist of one data segment for each tree in the current system and are accessed by the eight-character treename associated with the data tree. The information contained in the treename files consists of (1) information concerning the lowest nodes currently in the tree, and (2) a table of mean vectors and packed covariance matrices containing one entry for each node in the data tree.

The parameters discussed below describe the format of files accessed under data tree names (Figure 3-10):

L1 The number of classes within the data tree. This parameter defines the extent of the COVMEAN block described below and is maintained in the twelve (12) higher-order bits of word 1 of the LOWNODE block.

L2 The number of dimensions associated with this tree; maintained in the intermediate twelve (12) bits of word 1.

L3 The number of lowest nodes in the data tree.

L4 A 24-word array which is currently not used.

L5 A table of data class names which represent the variable portion of the data filenames under the data tree. Each of the 72 entries occupies one computer word.

The following parameter list describes an entry within the COVMEAN block of the treename file. The number of entries currently being maintained is found in parameter L1 of the LOWNODE block.

CM1 The four-character nodename associated with the data class represented by the entry.

CM2 The number of data vectors contained in the data class.

CM3 A table of floating point numbers representing the mean vector for the data class. The table contains one value for each measurement associated with the data tree (parameter L2 in the LOWNODE block of the file).

CM4 A table of floating point numbers representing the packed covariance matrix for the data class.

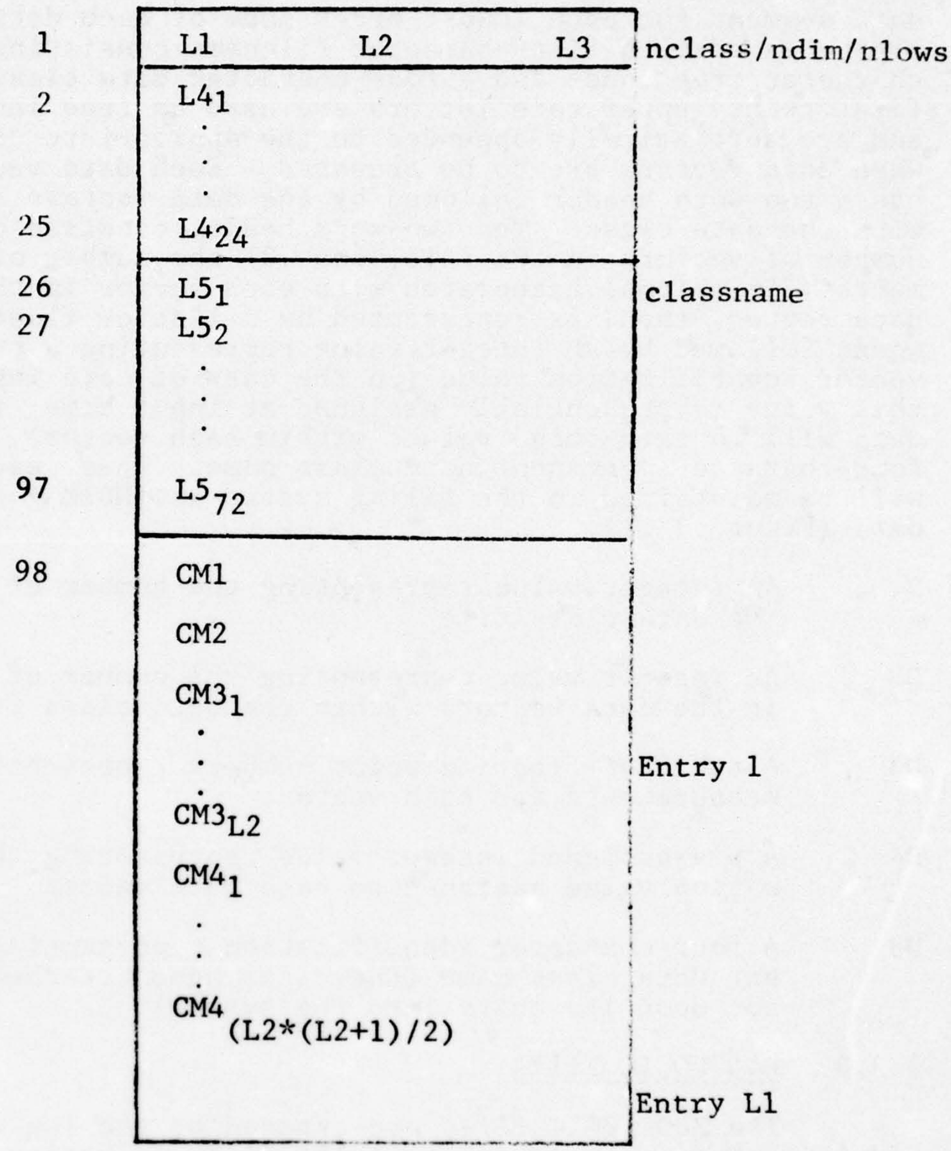


Figure 3-10: treename File Format

3.3.2. The DATACLASS Files

The DATACLASS Files are created during data input, structure analysis partition or transformation. They consist of one data segment for each lowest-order node of each data tree and are accessed by a five-character filename consisting of a one-character tree index and a four-character data class name. The first twenty upper case letters are used as tree index characters and are automatically appended to the appropriate data class name when data vectors are to be accessed. Each data vector file has a two-word header followed by the data vectors associated with the data class. The two-word header consists of: (1) the number of vectors in the file, and (2) the number of measurements (dimensions) associated with each vector in the file. Each data vector, then, is represented by a list of floating point words followed by an integer value representing a pre-assigned vector identification value (in the case of card input data, this value is sequentially assigned at input time; tape input data will contain these values within each vector), and by a four-character permanent data class name. Thus, each vector will be maintained in the filing system as (NDIM + 2) words of data (Figure 3-11).

D1 An integer value representing the number of vectors in the data class file.

D2 An integer value representing the number of dimensions in the data vectors within the data class file.

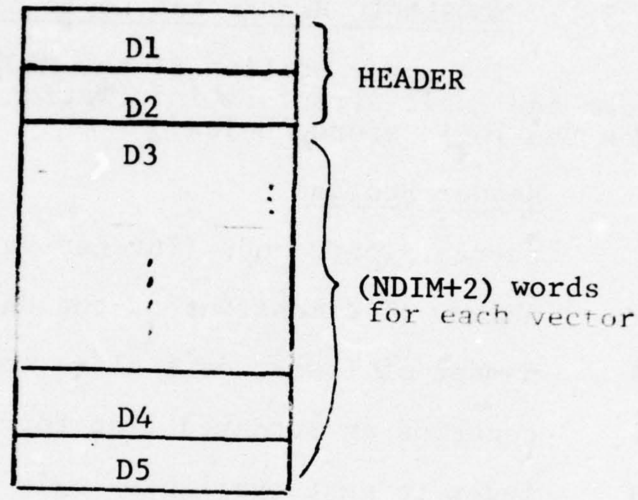
D3 A table of floating point numbers representing the data measurements for each vector.

D4 A pre-assigned integer value representing the identification value assigned to each data vector.

D5 A four-character identification representing a permanent data class name (the class name attached to the vector upon its entry into the system).

3.3.3. MOOSLOGIC Files

The MOOSLOGIC files are created by the logic design modules and are accessed by a valid eight-character treename concatenated with the appropriate data class name and augmented with the characters "logic." The various logic designs available to the MOOS user have been described in Section 2. Data logic files are described below in several sections since the format varies for each logic type:



D1 - - - number vectors
D2 - - - number dimensions
D3 - - - DATA measurements
D4 - - - ID. number
D5 - - - DATA class name

Figure 3-11: Dataclass File Format

- o header and logic structure block
- o pairwise logic block
- o grouped partition logic - one-space
- o grouped partition logic - two-space
- o grouped Boolean logic
- o nearest mean vector logic
- o independent Boolean reject logic

3.3.3.1. MOOSLOGIC Header and Logic Structure Block

The upper portion of the MOOSLOGIC file contains the header and logic structure information required to identify and index the logic stored below.

A. Header Section

SH1 current logic node (integer value)

SH2 number of dimensions of the data class (integer value)

SH3 number of lowest data class nodes (integer value)

SH4 contains an outdated flag formerly used by sln

SH5 index to next available logic block (integer value)

SH6 last logic node to have been defined

SH7 flag setting for creation of group logic. If set = 1 the logic to be created is group logic

B. (for $1 \leq i \leq 128$) Logic Node Descriptions

$SN1_i$ - (9 bits) logic type of node i
if 0 - undefined
1 - pairwise
2 - group
3 - nearest mean vector

$SN2_i$ - (9 bits) superior logic node number

$SN3_i$ - (9 bits) number of logic nodes below node i at next level

$SN4_i$ - (9 bits) 8th bit is set if this node has been modified by mod10. 9th bit is set if this is a lattice logic node

$SN5_i$ - (72 bits) a parallel bit map to the data class section of the structure part of the MOOS logic file (SC1 and SC2)
if 1 - corresponding class is present
0 - not present

$SN6_i$ - (36 bits) relative index from the beginning of the file to the logic block for this logic node.

$SN7_i$ - (36 bits) relative index from the beginning of the file to the reject strategy logic block for this logic node. 0 - means no reject strategy defined.

$SN8_i$ - (36 bits - 4 characters) reassociated data class name (only applicable to lowest nodes)

C. (for $1 \leq i \leq SH3$) Data Class Descriptions

$SC1_i$ - (36 bits) data class name (four characters)

$SC2_i$ - (36 bits) a table parallel to SC1 giving the a priori probability of a vector from $class_i$.

D. Cost Matrix

SCM. SH3* (SH3+1) array of costs associated with misclassifications or reject

HEADER PART OF STRUCTURE SECTION OF MOOSLOGIC FILE

1	SH1	current logic node
2	SH2	number of dimensions
3	SH3	number of lowest nodes
4	SH4	not used
5	SH5	index to next logic block
6	SH6	index to last logic defined
7	SH7	
32	not used	

NODE PART OF STRUCTURE SECTION OF MOOSLOGIC FILE

33	$SN1_1$ | $SN2_1$ | $SN3_1$ | $SN4_1$	type/sup. node/no. of nodes below/flags
34 35	$SN5_1$	classes at node bit map
36	$SN6_1$	index to logic block for this node
37	$SN7_1$	index to reject strategy block
38	$SN8_1$	reassociated data class name
39 40 41	⋮	
	$SN1_{128}$	
	$SN5_{128}$	
	$SN6_{128}$	
799	$SN7_{128}$	
800	$SN8_{128}$	

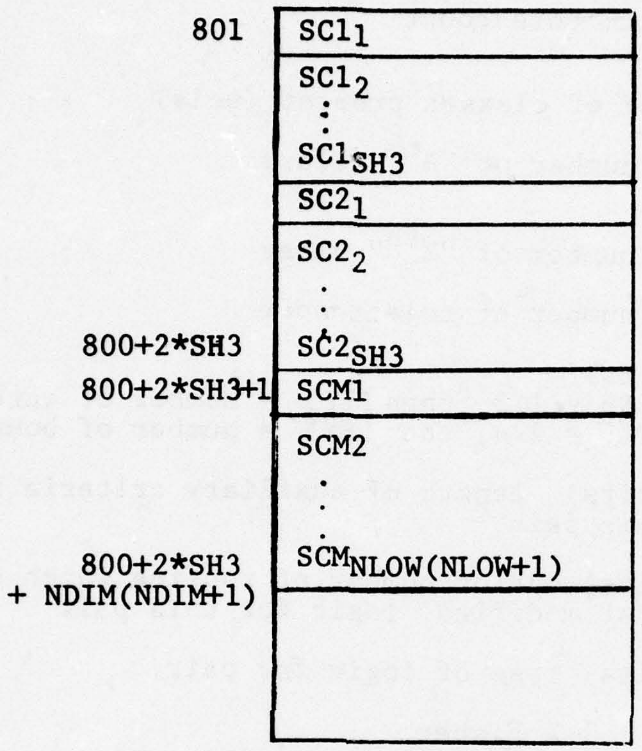


Figure 3-12: <u>Format for Header and Structure of MOOSLOGIC File</u>

3.3.3.2 Pairwise Logic for MOOSLOGIC Files

Pairwise logic contains one section of logic for each pair of data classes for which discrimination is desired.

LPR1a	minimum vote count
LPR1b	number of classes present (ncls)
LPR2(1)	node number of "A^{th}" class
⋮	
LPR2(ncls)	node number of "Z^{th}" class
LPR2(ncls+1)	node number of reject node
LPR3(1)	(3 bits) if LPR6 = 1/3, then LPR3 = number of thresholds if LPR6 = 2/4, then LPR3 = number of boundaries
LPR4	(15 bits) length of auxiliary criteria block of this pair
LPR5	(9 bits) option number of routine which created, or last modified, logic for this pair
LPR6	(9 bits) type of logic for pair: 1 - Fisher 2 - optimal discriminant plane 3 - any one-space 4 - any two-space 5 - linguistic logic
LPR8 and LPR9	(9 bits) if a class pair is denoted by "A/B", (9 bits) then LPR9=node number of "A"; LPR8= node number of "B"
LPR10	(18 bits) index to criteria block
LPR11	index to auxiliary criteria block (if any) for this pair
LPR12	(ndim words) coefficients of Fisher discriminant
LPR13	(ndim words) coefficients of line perpendicular to Fisher direction
LPR14	(5 words) five thresholds along Fisher direction
LPR15	logic blocks for modified pairwise logic

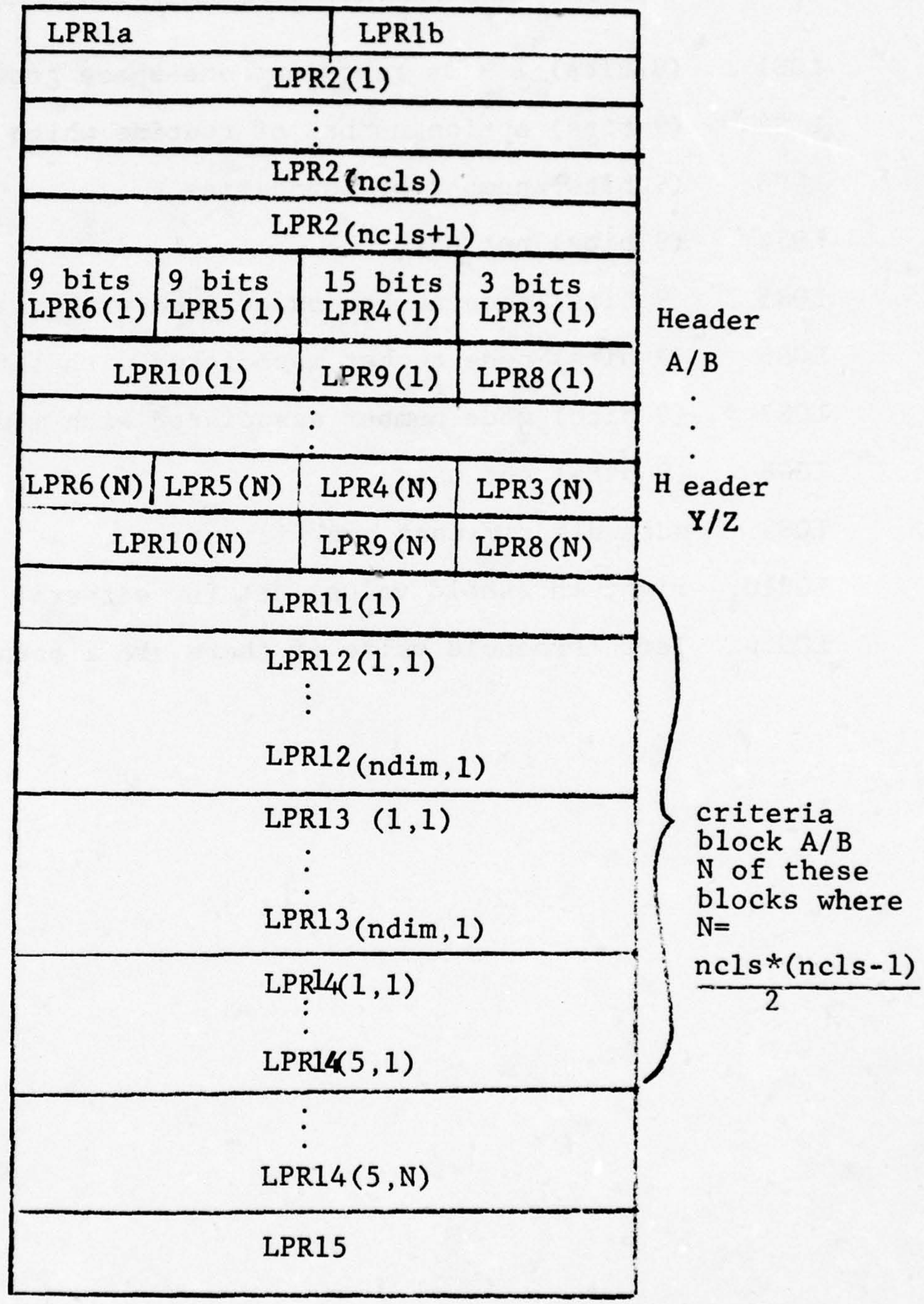


Figure 3-13: <u>Pairwise Logic Format for MOOSLOGIC File</u>

3.3.3.3. Logic Block Format for One-Space Group Logic

LOS1	(9 bits) 2 - to represent one-space group
LOS2	(9 bits) option number of routine which created logic
LOS3	(9 bits) number of boundaries
LOS4	(9 bits) not used
LOS5	(9 bits) node number associated with right-most region
LOS6	(9 bits) node number associated with left-most region
LOS7	(9 bits) node number associated with middle region
LOS8	(9 bits) not used
LOS9	ndim discriminant coefficients
$LOS10_1$	right threshold value, set for either 1 or 2 boundaries
$LOS10_2$	left threshold value if there are 2 boundaries

LOS1 | LOS2 | LOS3 | LOS4	2/option no./NBNDS/N.U.
LOS5 | LOS6 | LOS7 | LOS8	right-most/left-most/middle/N.U.
$LOS9_1$ . . . $LOS9_{ndim}$	ndim coefficient
$LOS10_1$	threshold right boundary
$(LOS10_2)$	threshold left boundary if it exists

Figure 3-14: Group Logic - One-Space Format for MOOSLOGIC Files

3.3.3.4 Logic Block Format For 2 Space Group Logic

LGP1	-	(9 bits)	-	1 - to represent plane-type logic.
LGP2	-	(9 bits)	-	option number of routine which created this logic
LGP3	-	(9 bits)	-	number of boundaries drawn
LGP4	-	(9 bits)	-	not used
LGP5	-	(9 bits)	-	logic node number associated with excess region
LGP6	-	(9 bits)	-	logic node number associated with convex side of boundary 1
LGP7	-	(9 bits)	-	logic node number associated with convex side of boundary 2
LGP8	-	(9 bits)	-	not used
$LGP9_1$	-	(36 bits)	-	number of line segments in boundary 1
$LGP10_{1,1,1}$ . . . $LGP10_{1,1,SH2}$				discriminant coefficients of first boundary for line segment 1 (floating point numbers)
$LGP11_{1,1}$			-	threshold value for line segment 1 of boundary 1
⋮				
$LGP10_{1,LGP9_1,1}$ . . . $LGP10_{1,LGP9_1,SH2}$				discriminant coefficients of boundary 1 for line segment $LGP9_1$
$LGP11_{1,LGP9_1,LGP9}$			-	threshold value for line segment $LGP9_1$ of boundary 1

only exists if 2 boundaries are drawn

$LGP9_2$	number of line segments in boundary 2
$LGP10_{2,1,1}$ . . . $LGP10_{2,1,SH2}$	coefficients of line segment 1, boundary 2
$LGP11_{2,1}$	threshold value of line segment 1, boundary 2
. . .	
$LGP10_{2,LGP9_2,1}$ . . . $LGP10_{2,LGP9_2,SH2}$	coefficients of line segment $LGP9_2$ boundary 2
$LGP11_{2,LGP9_2}$	threshold value for line segment $LGP9_2$, boundary 2

Note: The second boundary stored is always on the "excess" side of the first boundary stored. 1st and 2nd here do not refer to the order in which the boundaries were drawn.

LGP1	LGP2	LGP3	LGP4	1/opt.no./NBNDS/N.U.
LGP5	LGP6	LGP7	LGP8	excess NN / convex NN 1st / convex NN 2nd / NU

$LGP9_1$	N Lines 1st boundary
$LGP10_{1,1,1}$ . . . $LGP10_{1,1,NDIM}$	coefficient 1st line seg. boundary 1
$LGP11_{1,1}$	threshold 1st line boundaryl
$LGP10_{1,2,2}$ . . . $LGP10_{1,2,NDM}$	
$LGP11_{1,2}$ . . .	
$LGP10_{1,LGP9_1}$ . . . $LGP10_{1,LGP9_1,NDIM}$	coefficient last line seg. boundary 1
$LGP11_{1,LGP9_1}$	threshold last line seg. boundary 1
$LGP9_2$ (only if 2nd line seg.) . . .	same for boundary 2 (if it exists)

Figure 3-15: Group Logic - Two-Space Format For MOOSLOGIC File

3.3.3.5 Logic Block Format for Boolean Logic Block

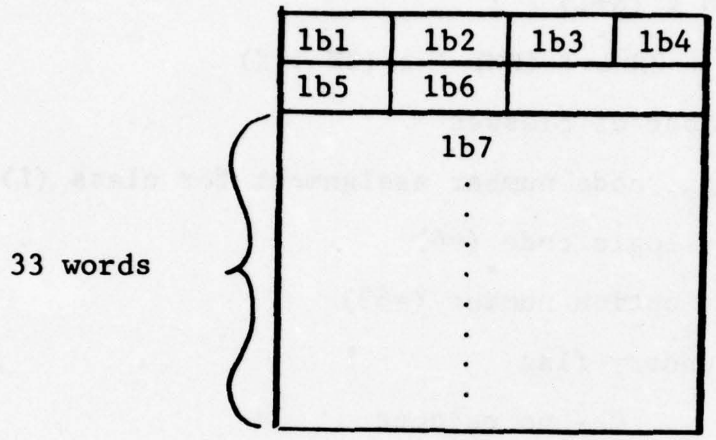


Figure 3-16: Grouped Boolean Logic Format for MOOSLOGIC Files

lb1	(9 bits)	"3" to represent Boolean logic
lb2	(9 bits)	option number of routine which created logic
lb3	(9 bits)	number of Boolean statements (always set to "1" by linglogc)
lb4	(9 bits)	not used
lb5	(9 bits)	node number of the logic node corresponding to the true "side" of the Boolean statement
lb6	(9 bits)	node number of the logic node corresponding to the false "side" of the Boolean statement
lb7	(33 words)	132 character (max) Boolean statement

3.3.3.6. <u>Nearest Mean Vector Logic Block</u>

N	=	number of dimensions
X	=	N x (N+1) / 2
Y	=	3 + 2NM1 + (NM1 - 1)(2N + X)
NM1	=	number of classes
NM2(i)	=	logic node number assignment for class (i)
NM3	=	nmv logic code (=6)
NM4	=	nmv option number (=63)
NM5	=	boundary flag 0 - no rejects 1 - reject
NM6	=	weight flag 0 - no weighting 1 - weighting vector 2 - weighting matrix
NM7(i)	=	reject value for class (i), equal to the square of the user's input value
NM8(i,j)	=	j^{th} component of mean of class i
NM9(i,j)	=	j^{th} component of weighting vector for class i (= inverse of j^{th} variance for class i)
NM10(i,j)	=	j^{th} component of packed inverse covariance matrix of class i

1	NM1
2	NM2(1)
	⋮
1+NM1	NM2(NM1)
2+NM1	NM2(REJECT)
3+NM	NM3 | NM4 | NM5 | NM6
4+NM1	NM7(1)
	⋮
3+2NM1	NM7(NM1)
4+2NM1	NM8(1,1)
	⋮
3+2NM1+N	NM8(1,N)
4+2NM1+N	NM9(1,1)
	⋮
3+2NM1+2N	NM9(1,N)
4+2NM1+2N	NM10(1,1)
	⋮
3+2NM1+2N+X	NM10(1,X)
	⋮
Y+1	NM8(NM1,1)
	⋮
Y+N	NM8(NM1,N)
Y+1+N	NM9(NM1,1)
	⋮
Y+2N	NM9(NM1,N)
Y+1+2N	NM10(NM1,1)
	⋮
Y+2N+X	NM10(NM1,X)

Figure 3-17: Nearest Mean Vector Logic Format for MOOSLOGIC Files

3.3.3.7. Closed Decision Boundary Logic Block

Cd1 - no. of classes

Cd2(1) - Cd2(ncls) - logic node numbers assigned to each class

Cd3 - reject logic node number

Cd4 - not used

Cd5(1) - Cd5(ncls) - indices to specific logic blocks (set up parallel to Cd2)

Cd6 - not used

Cd7 - 1 if user requested that a tree be created from "overlap" vectors, 0 otherwise.

Cd8 - logic type: 1 = hyperrectangular, 2 = hyperspherical, 3 = hyperellipsoid

Cd9 - basis vector type: 1 = coordinate, 2 = eigenvectors 3 = eigenvectors of the specific class

Cd10 - not used

Cd11(1) - Cd11(ndim) - set to 0 if the thresholds are based on the range of the data, 1 if they are user-defined

Cd12(1) - Cd12(ndim) - low thresholds along each measurement

Cd13(1) - Cd13(ndim) - high thresholds along each measurement

Cd14 - 0 for coordinate vectors or eigenvectors of a specific class basis vectors types. Index to basis vectors for eigenvectors of entire data set basis vectors type

Cd15 - basis vectors (not stored for coordinate vectors basis type)

Cd16 - center vector type: 1 = mean of the class, 2 = median of the class, 3 = user-defined

Cd17 - not used

Cd18(1) - Cd18(ndim) - center vector

Cd19 - set to 0 if the radius is based on the range of the data, 1 if the radius is user-defined

Cd20 - radius squared of hypersphere

Cd21 - axis length type: 1 = Mahalanobis (axes = eigenvalues), 2 = range of data, 3 = user-defined

Cd22(1) - Cd22(ndim) - axis lengths

Cd23 - "C" value type: 0 = based on the range of the data, 1 = user-defined. (the "C" value is a quantity analogous to the radius of a hypersphere)

Cd24 - "C" value

Cd25 - weighting matrix

Header of closed decision boundary logic block

(ncls = No. of classes) 1	Cd1	ncls
2	$Cd2_1$	logic node numbers
1 + ncls	$Cd2_{ncls}$	
2 + ncls	Cd3	reject node no.
3 + ncls	Cd4	
4 + ncls	$Cd5_1$	indices to logic blocks for each class
3 + 2*ncls	$Cd5_{ncls}$	
4 + 2*ncls	Cd6	
5 + 2*ncls	Cd7	
		specific logic blocks for each of the classes

Hyperrectangular closed decision boundary logic block

1	Cd8	type = 1
2	Cd9	
3	Cd10	
4	$Cd11_1$	repeated for each dimension (high and low thresholds)
5	$Cd12_1$	
6	$Cd13_1$	
	⋮	
1 + 3*ndim	$Cd11_{ndim}$	
2 + 3*ndim	$Cd12_{ndim}$	
3 + 3*ndim	$Cd13_{ndim}$	
4 + 3*ndim	Cd14	
5 + 3*ndim	$Cd15_{1,1}$	basis vectors
	⋮	
	$Cd15_{1,ndim}$	
	⋮	
	$Cd15_{ndim,ndim}$	

Hyperspherical closed decision boundary logic block

1	Cd8	type = 2
2	Cd16	
3	Cd17	
4	$Cd18_1$	center vector
	⋮	
	$Cd18_{ndim}$	
4 + ndim	Cd19	
5 + ndim	Cd20	radius

Hyperellipsoid closed decision boundary logic block

1	Cd8	type = 3
2	Cd16	
3	Cd17	
4	$Cd18_1$	center vector
	⋮	
	$Cd18_{ndim}$	
4 + ndim	Cd21	
5 + ndim	$Cd22_1$	axis lengths
	⋮	
	$Cd22_{ndim}$	
5 + 2*ndim	Cd23	
6 + 2*ndim	Cd24	"C" value
7 + 2*ndim	$Cd25_{1,1}$	weighting matrix
	⋮	
	$Cd25_{1,ndim}$	
	⋮	
	$Cd25_{ndim,ndim}$	

3.3.3.8. Independent Boolean Reject Logic Block

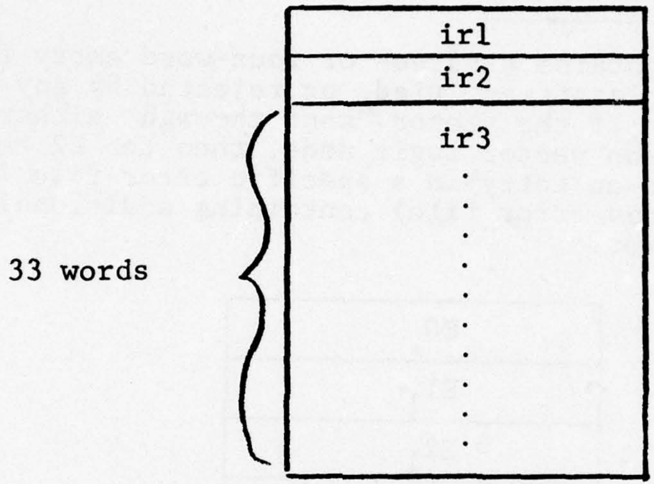


Figure 3-18: Boolean Reject Logic Format for MOOSLOGIC Files

Logic block format:

ir1	(1 word)	not used.
ir2	(1 word)	set to logic node number of first (if there are more than one) logic node to use this Boolean reject statement.
ir3	(33 words)	132-character (max) Boolean reject statement.

3.3.3.9. Logic Error Files

Any or all of the following files are created by the logic evaluation routines to store incorrectly classified vectors.

"error_f" file format

"error_f" contains a three- or four-word entry for each vector which is misclassified, tied, or rejected by any logic evaluation routine. If the vector "went through" either a pairwise or a nearest mean vector logic node, then the E2 entry exists and points to an entry in a specific error file (either pair_error_file or nmv_error_file) containing additional information about the error.

E0
$E1_1$
$E2_1$
$E3_1$
$E4_1$
⋮
E1 (no. of errors)
E2 (no. of errors)
E3 (no. of errors)
E4 (no. of errors)

E0 true class

E1 error type: 1 - group (including one-space, two-space, and Boolean)
2 - pairwise
3 - nearest mean vector
4 - independent reject

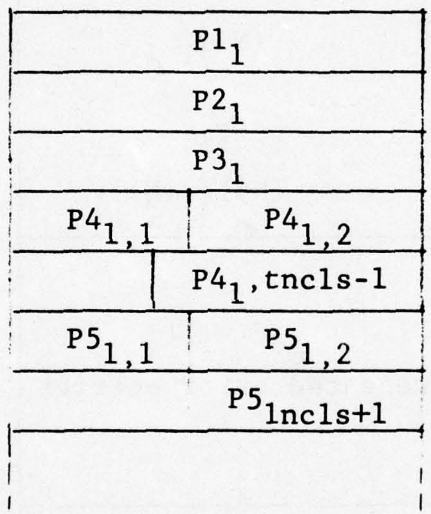
E2 The E2 entry exists only for types 2, 3, and 5. For type 2 it points to an entry in the pair_error_file; for type 3 it points to an entry in the nmv_error_file; for type 5 it points to an entry in the box_error_file.

E3 The ID number of the vector in error

E4 The logic node number of the logic node to which the vector was assigned

"pair_error_file" file format

pair_error_file contains specific information regarding vectors which are misclassified, tied or rejected at pairwise logic nodes. It is generated by the internal subroutine pairwise_logic and the evaluation routines.



this block repeated for each misclassified vector

lncls number of classes in the logic tree

tncls number of data classes being evaluated

P1 logic node number of true class

P2 set to minimum vote count threshold by evaluation routines. If the error was the result of a tie, the number of tied class counters is inserted by pairwise_logic. If the error was not the result of a tie, P2 is set to zero.

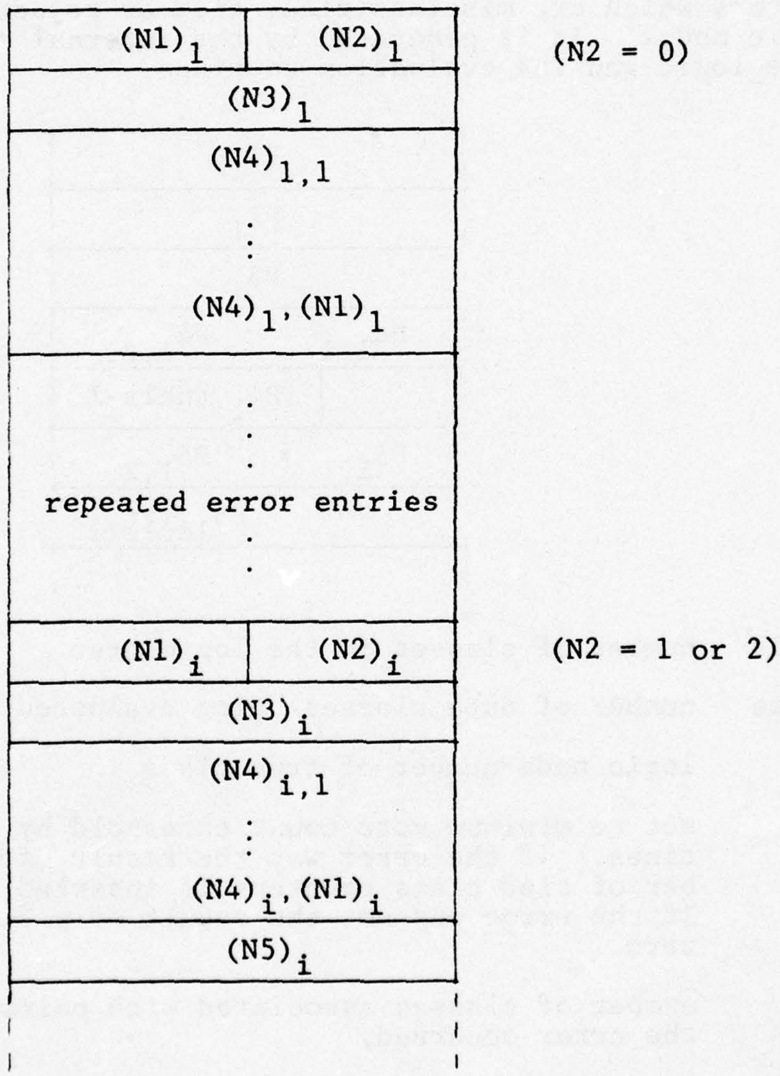
P3 number of classes associated with pairwise node where the error occurred.

P4 (9 bits) (lncls-1) entries containing the logic node number to which the vector was assigned in each decision box involving the true class.

P5 (12 bits) (lncls+1) entries containing the vote counts at each logic node.

"nmv_error_file" format

nmv_error_file contains specific information regarding vectors which are misclassified, tied or rejected at nearest mean vector logic nodes. It is generated by the internal subroutine nmv_logic and the evaluation routines. The format varies somewhat depending on the cause of the error. An example of each type is given.



(N1) (18 bits) number of ties

(N2) (18 bits) set to logic node number of the true class by evaluation routines. If the error was not the result

of a tie or reject, N2 is set to 0 by nmv_logic. If the error was caused by a reject, boundary N2 is set to 1. If the error was a tie, N2 is set to 2.

(N3) distance to the true class

(N4) an array of four character class names. If N2 is 1, these names represent the closest classes to the vector. If N2 is 2, these names are the classes from which the vector is equidistant.

(N5) distance to the closest class(es) (N2 = 0) or tied classes (N2 = 2)

"box_error_file" format

box_error_file contains specific information about vectors which are misclassified, rejected, or assigned to more than one class at closed decision boundary logic nodes. It is generated by internal subroutine box_logic and the evaluation routines.

B1	B2
B3	
B4	

This four word block is repeated for each vector which is not correctly classified

B1 (18 bits) set to 1 if the vector was an error; 2 if the vector was rejected; 3 if the vector fell into more than one hyperregion

B2 (18 bits) logic type: 1 = hyperrectangular, 2 = hyperspherical, 3 = hyperellipsoid

B3 (72 bits) bit map. If a bit is set, the vector fell into the hyperregion associated with the class which corresponds to the bit (the name of the class is found in the SC1 section of the logic file)

B4 (36 bits) not used

3.4. STANDARD PERMANENT SYSTEM DIRECTORY FILES AND DIRECTORIES

3.4.1. "option_file" File

The "option_file" file is maintained in the system default working directory; it contains a list of all variable options (in two sections of 256 slots each) and a pointer section detailing the menus to be displayed for each MOOS function. Figure 3-17 describes the "option_file" file, utilized by the internal system program "option" to generate the display menus. Programmer aids option$list and option$insert utilize the "option_file" file to list and modify their contents, respectively.

01 The MOOS function names section contains a list of the currently implemented MOOS functions in order of slot number. The following general set of blocks within this section has been assigned (Section 4.1. contains a list of MOOS function numbers):

Slot Number	MOOS Application
1 - 19	Data Input
20 - 59	System Utility
60 - 127	Distribution Free Logic Design
128 - 195	Transformations and Measurement Evaluation
196 - 255	Structure Analysis

Options added (via "option$insert" type 1 operations) to this section of the file are directly associated with an entry in the menu index portion of the file (03).

02 The Utility Function names section contains a list of the currently implemented Utility Functions which may appear on menus, but which do not have a menu directly associated with them. Names added to this section are assigned to the first open block found within the 02 section.

03 The Menu Index Section lists up to 12 slot numbers (in either the 01 or 02 blocks) to be displayed as a menu associated with a parallel entry number in the 01 block.

	1	9'10	18'19	27'28 36	
1			null		MOOS function name
2			null		
3			01_1		
4			01_1		
5			01_2		
6			01_2		
			. . .		
511			01_{255}		
512			01_{255}		
513			02_1		Utility function name
514			02_1		
			. . .		
1023			02_{256}		
1024			02_{256}		
1025	$03_{1/1}$	$03_{1/2}$	$03_{1/3}$	$03_{1/4}$	Menu Index
1026	$03_{1/5}$ . . .				
1027	$03_{1/9}$			$03_{1/12}$	
			. . .		
1787	$03_{255/1}$			$03_{255/4}$	
1788	$03_{255/5}$ . . .				
1789	$03_{255/9}$ . . .			$03_{255/12}$	

Figure 3-19: The "option_file" File Format

3.4.2. "trandata" Directory

The MOOS system is a multi-user environment providing each user with exclusive access to the data in his process directory. Many times a user has a need to reference a data tree generated by another user. To accommodate this need, the directory "trandata" has been created with all users granted unrestricted access to the resident data trees.

Two utility functions (savec, restorec) have been provided for the purpose of (1) depositing (savec) a tree in "trandata", and (2) restoring (restorec) a tree from "trandata" to the user's process directory. A third utility function (remtree) provides for the removal (deletion) of a tree from "trandata".

There are two types of segments in "trandata". The first is a data reference name which is automatically inserted or removed when the user executes savec or remtree. The second segment is a file called "seg_o_trees"; this file acts as a program directory of all stored data tree segments in "trandata" and is also automatically updated in response to the user commands (see Section 3.5.2.).

3.4.3. "olpars" archive

The "olpars" archive contains the source programs for all MULTICS/OLPARS system programs. Storing source code in this manner provides for ease of modification of any system option or internal routine. Since only those routines which have been changed must be recompiled, system modifications and additions are swift and simple to implement.

3.5 . STANDARD PERMANENT USER DIRECTORY FILES AND DIRECTORIES

3.5 .1. "saved_trees" Directory

In many instances a user may not complete his analysis of a data set (tree) in one computer session. Normally, data trees reside in the users process directory. However, this is a temporary directory and its contents are lost when MULTICS is shut down or the user logs out. To give the user the ability to save a tree(s) from one computer session to another a permanent directory called "saved_trees" has been incorporated into MOOS.

"saved_trees" is a subdirectory that is added to the user's own directory, therefore each user would have his own "saved_trees." The security of all data is maintained since only the creating user has access to his copy of "saved_trees."

There are three types of entries in this directory; one is the lowest node data and the others are tables called "seg_o_trees" and "seg_o_logic."

- o lowest node - as copies of the various vector groupings (classes) are made in "saved_trees" an appropriate entry is made to allow future references to those vector groups.
- o "seg_o_trees" - see Section 3.5.2.
- o "seg_o_logic" - a list of logic files stored within "saved_trees"

3.5 .2 "seg_o_trees" File

This table contains the name of each tree and its associated nodes as they appear in "saved_trees" or "trandata." This table is built automatically by the program "s_p" when the operator executes the savec or save options.

It should be noted that "saved_trees" is only a temporary 'overnight' storage for data trees. All manipulation of data is performed on trees that reside in the user's process directory. For example, if a user were to copy a tree into "saved_trees" and then perform a partition of the tree in the process directory, that subsequent operation would not appear in "saved_trees."

Naming Convention

As it appears in "sysdata", each tree has an eight-character name and each node a four-character name prefixed by the tree symbol. The eight-character tree name is checked for uniqueness (an alternate is requested if it is not unique) and inserted in "seg_o_trees" along with all four-character node names associated with that tree. Each lowest node name is generated by concatenating the eight-character tree name with the four-character node name. Thus, "nodl" in the "irisdata" tree will become lowest node entry "irisdatanodl."

"seg_o_trees" format

0	K1	count of entries	
1,2	E1	tree name	tree entry
3	E2	count of nodes	
4,4+n	E3	node name	

Word 0 is a count of trees currently stored in "seg_o_trees"

Words 1 through 4+n constitute a tree entry and will be repeated for each tree stored.

Words 1 and 2 contain an eight-character ASCII tree name

Word 3 contains the count of lowest nodes in the tree

Words 4 - 4+n contain a list of four-character ASCII node names.

3.5.3. saved "vectors" file

The "vectors" file is a file of stored projection vectors in the user's working directory (see subroutines vec$save, vec$list, vec$hall, vec$all, vec$del) and is accessed by arbitrary vectors projection operations.

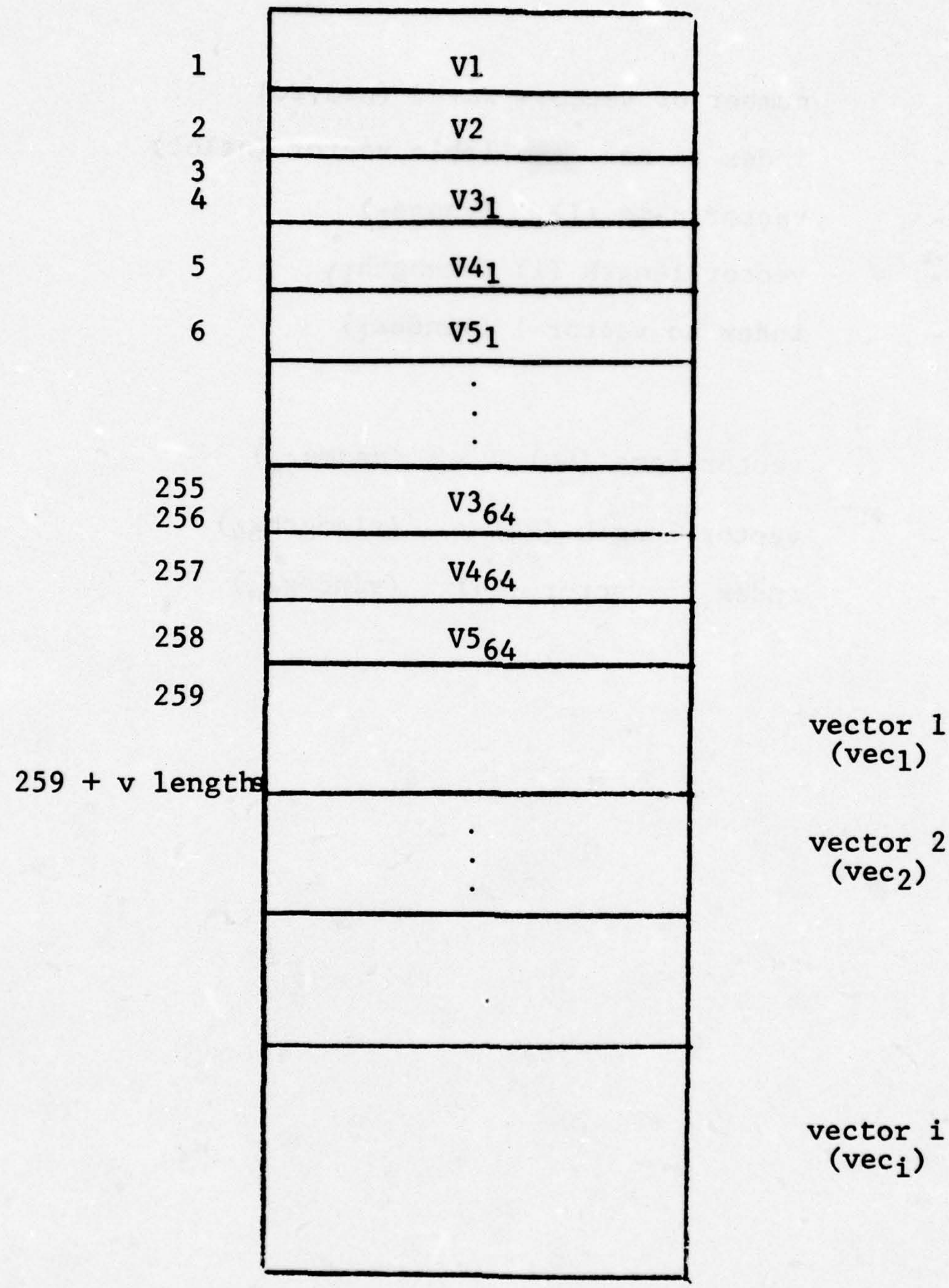


Figure 3-20: "vectors" File Format

V1	-	number of vectors saved (nsaved)
V2	-	index to next available vector (nslot)
$V3_1$	-	vector name (1) ($vname_1$)
$V4_1$	-	vector length (1) ($vlength_1$)
$V5_1$	-	index to vector 1 ($vindex_1$)
.		
.		
.		
$V3_{64}$	-	vector name (64) ($vname_{64}$)
$V4_{64}$	-	vector length (64) ($vlength_{64}$)
$V5_{64}$	-	index to vector (64) ($vindex_{64}$)

3.5.4. "seg_o_logic" file

The "seg_o_logic" file contains a list of all references under which MOOS logic files have been saved in the "saved_trees" directory (see subroutines log$save, log$rstr, log$delt, log$list).

The "seg_o_logic" file acts in conjunction with the "saved_trees" directory as a storage area for MOOS logic files that the user may wish to save from day to day. It should be noted that, as with any saved trees, if the user updates his current version of a logic file in the process directory, no change is made to the saved logic.

1	SL1
2	SL2(1)
3	SL2(1)
4	SL3(1)
5	SL2(2)
6	SL2(2)
7	SL3(2)
	.
	.
	.
(SL1-1)*3+2	SL2(SL1)
(SL1-1)*3+3	SL2(SL1)
(SL1-1)*3+4	SL2(SL1)

SL1 - number of saved logic files

SL2(i) - eight-character tree name reference for saved logic file(i)

SL3(i) - four-character node name reference for saved logic file(i)

SECTION 4

MOOS PROGRAM DOCUMENTATION

4.1 INTRODUCTION

This section, along with Section 3, is not aimed toward the average MOOS User, but is for system programmers who might be interested in modifying or expanding the MOOS system.

4.2 MOOS PROGRAM GENERATION

The following procedure will permit a programmer to enter a source program via card inputs through the MULTICS I/O DEAMON. The rational for this procedure is to allow maximum MULTICS time to be used for program debug and execution. This technique affords an off-line generation of a source deck on an 029 keypunch which will subsequently be read by the MULTICS I/O DEAMON.

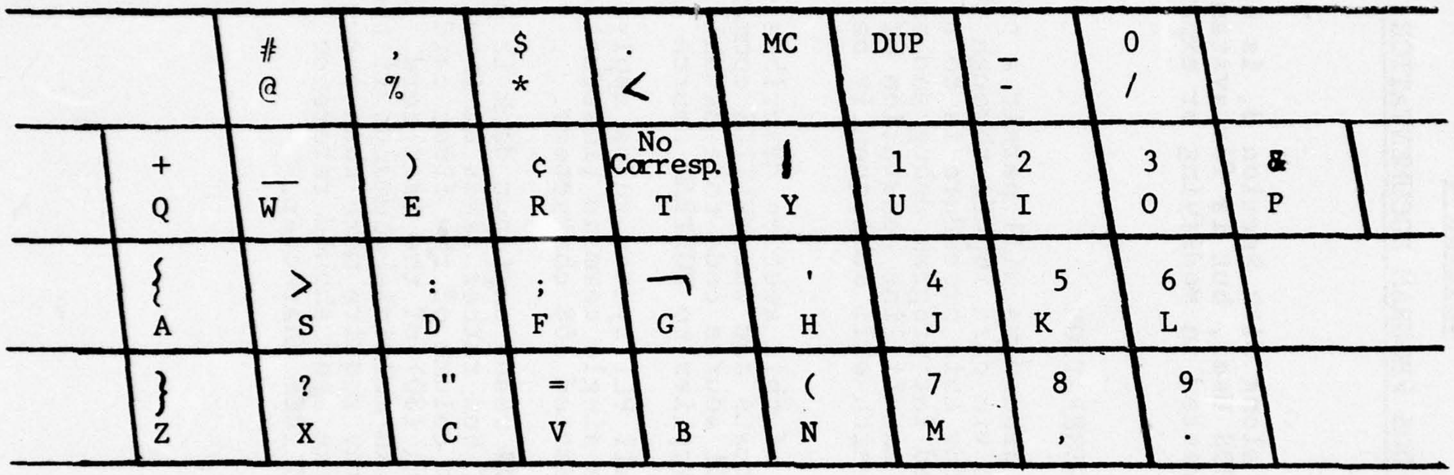
The remainder of this section describes control card/terminate card formats and the MULTICS commands required to convert the PL1 source code to a usuable form. Of course, this section applies to FORTRAN source code as well.

- o all PL1 syntax rules apply
- o a single command (instruction) may not exceed 168 characters.

When the DEAMON reads an input deck it does not distinguish between cards, but rather reads an input stream starting with the first column of the first card and ending with the last column (80) of the last card. For this reason, any card may contain multiple commands if the programmer chooses, or a command may require more than one card. When using the 029 keypunch the user should reference Figure 4-1 for the appropriate MULTICS character.

Figure 4-1

MULTICS KEYBOARD



Discrepancies: + { } < _ | >: ¬ ' ? " =

Run Deck Structure

The run deck will be in the following format:

SYSTEM CONTROL CARDS
PROGRAMMER SOURCE DECK
SYSTEM CONTROL CARDS

Run Procedure

Assemble the input deck and give it to the MULTICS operator requesting the DEAMON be used to read the deck. The operator should inform you that the read was successful.

4.2.1 EXCESS MEASUREMENT MODE

An excess measurement mode has been implemented for entering data sets with greater than the system limit of 100 dimensions. A small subset of MOOS functions and utility functions may be run on data sets with up to 250 dimensions using this feature. The following is a list of MOOS programs which are limited to 250 dimensions by declarative statement only. If a program is an excess measurement program but is not listed below, there is no inherent limit on dimensionality. A more complete list of excess measurement mode programs available for MOOS users may be found in Section 1.

clusscat
crdv
dataprnt
dscrmeas
measxfrm
npcos
probconf
ss
tapinput
transgen
trnsform

4.3 MOOS PROGRAM DESCRIPTIONS

The following pages describe the programs currently implemented under MOOS. Each program description includes a complete description of input and output parameters and file settings, a functional description of the program and a high level flow chart.

Programs are categorized as MOOS Functions (routines called by the user from his console), internal subroutines, and programmer aid routines.

MOOS Functions are further subdivided into Major MOOS Functions and MOOS Utility Functions. Table 4-1 is a listing of Major MOOS Functions in order of Major MOOS Function number.

Internal subroutines are transparent to the user and cannot be called from the console directly but are called by MOOS Functions.

Programmer aid routines are callable from the console and are designed for system debugging and maintenance.

Table 4-2 is an alphabetic listing and page number index of all routines. Routines with multiple entry points are not separately indexed.

Table 4-1

Numerical Index of Major MOOS Functions

Number	Function	Number	Function
1	crdinput	68	ardg$1d2
2	tapinput	69	asdg$1d2
3	creatree	70	eigv$1d2
4	mergmeas	71	fshp$1d2
19	crrandts	72	gndv$1d2
21	dsubstrc	81	arbv$1d1
26	comnod	82	crdv$1d1
27	listlogc	83	ardg$1d1
28	deletlog	84	asdg$1d1
29	chngname	85	eigv$1d1
30	chngaprb	86	gndv$1d1
40	dvectors	95	forteval
50	deletnod	96	nmvmod
61	lingrjct	97	pairmod
62	linglogc	98	closedcn
63	nmv	99	closemod
64	logicevl	128	eigentrn
65	fisher	130	normxfrm
66	arbv$1d2	141	probconf
67	crdv$1d2	142	dscrmeas
		143	features
		196	arbv$sa2
		197	crdv$sa2
		198	lingpart
		199	ardg$sa2
		200	asdg$sa2
		201	eigv$sa2

Number	Function
202	nlm
203	fshp$sa2
204	gndv$sa2
211	arbv$sa1
212	crdv$sa1
213	ardg$sa1
214	asdg$sa1
215	eigv$sa1
216	gndv$sa1
255	*********

Table 4-2

Alphabetic Listing and Page Number Index of all Routines

Name	Description	Page
aevs	computes eigenvectors of a non-symmetric matrix	4-16
ans	retrieves answer to a "yes/no" question	4-17
anything	list current MOOS functions	4-18
append	add a data class from other existing data sets	4-20
arbv	arbitrary vector projection	4-22
ardg	arbitrary discriminant grouping projection	4-25
asdg	assigned discriminant grouping projection	4-28
binwidth	one-space display modification	4-33
boologic	temporary boolean logic evaluation routines	4-35
box_logic	closed decision boundary logic evaluation	4-36
boxprogram	generates closed decision boundary code for FORTRAN subroutine logic	4-37
cdblogic	closed decision boundary logic information printout	4-38
cdefault	change default values for data projections	4-39
cdisplay	cluster/scatter display switching	4-41
checkp	extract input parameters	4-42
chngaprb	a priori probability modification	4-43
chngname	tree name or node name modification	4-47
cl_restruct	nonlinear map data set restructuring	4-49
cleartree	data tree removal form exclusive user storage	4-51
closedcn	create closed decision boundary logic	4-52
closemod	modify a closed decision boundary logic node	4-53
close_ut	utility routine used by closemod	4-54

<u>Internal Subroutine Name</u>: aevs

<u>Calling Sequence</u>: call aevs (nv, nf, c, e)

<u>Input Parameters</u>:

<u>nv</u>	fixed (35) order of input matrix <u>a</u>.
<u>nf</u>	fixed (35) maximum no. of eigenvectors to be extracted.
<u>c</u>	float minimum eigenvalue to be extracted.
<u>a</u>	(100,100) float external static square input matrix, destroyed in process.

<u>Output Parameters</u>:

<u>nf</u>	fixed (35) no. of eigenvalues extracted which exceed minimum value <u>c</u>.
<u>v</u>	(100,100) float external static output array of eigenvectors (columns)
<u>e</u>	(100) float array of eigenvalues.

<u>Program Description</u>:

aevs is a direct translation to PL/1 of a FORTRAN subroutine designed to extract the eigenvectors of a non-symmetric square matrix. The technique is iterative, and the no. of iterations has been arbitrarily set at 50. The no. of iterations could be changed to improve accuracy or to obtain greater computational speed.

Internal Subroutine Name: ans

Calling Sequence: call ans (b)

Output Parameters:

b	bit(1)	b = "1" b if the answer is yes; b = "0" b if the answer is no.

Program Description:

ans reads the answer to a "yes/no" question, checks the answer to determine if it is correctly spelled, and returns. The Boolean variable b is set only if the answer is yes.

<u>Utility Function Name</u>: anything

<u>Calling Sequence</u>: Type in "anything"

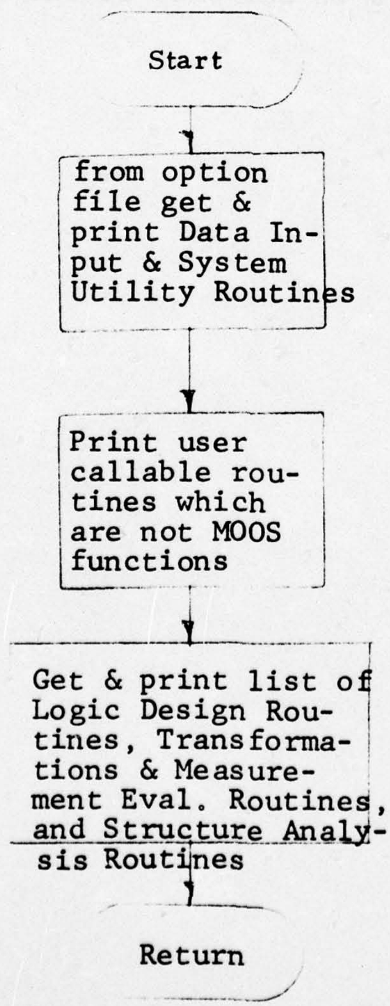
<u>Input File Setting</u>: None

<u>Output File Setting</u>: None

<u>Program Description</u>: This program prints to the user currently implemented MOOS functions, and user callable routines.

<u>Flow Chart</u>:

anything



Internal Subroutine Name: anything$getopt

Calling Sequence: call anything$getopt (j, ptr)

Input Parameters:

j — The number of the option to be printed [fixed bin (35)]

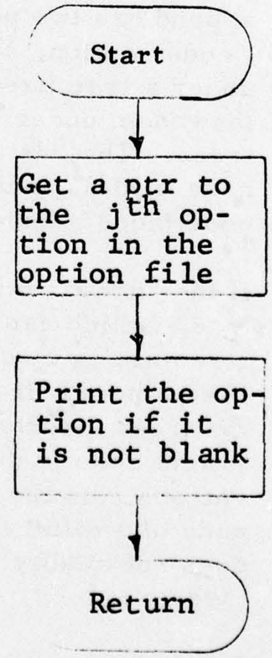
ptr — pointer to the beginning of the option file [ptr]

Input File Settings: The option file must exist.

Program Description: "anything$getopt" is the routine called from "anything" which retrieves the j^{th} option from the option file. It also formats the option for printing.

Flow Chart:

anything$getopt



Utility Function Name: append

Calling Sequence: Type in "append (treel, nodel, tree2, node2, [newnode])"

Input Parameters:

- treel - tree from which data is coming from.
- nodel - lowest node from which data is coming from.
- tree2 - tree to which data is being added.
- node2 - lowest node to which data is being added to or intermediate node to which data is being added under.
- newnode - name of new node being created (optional).

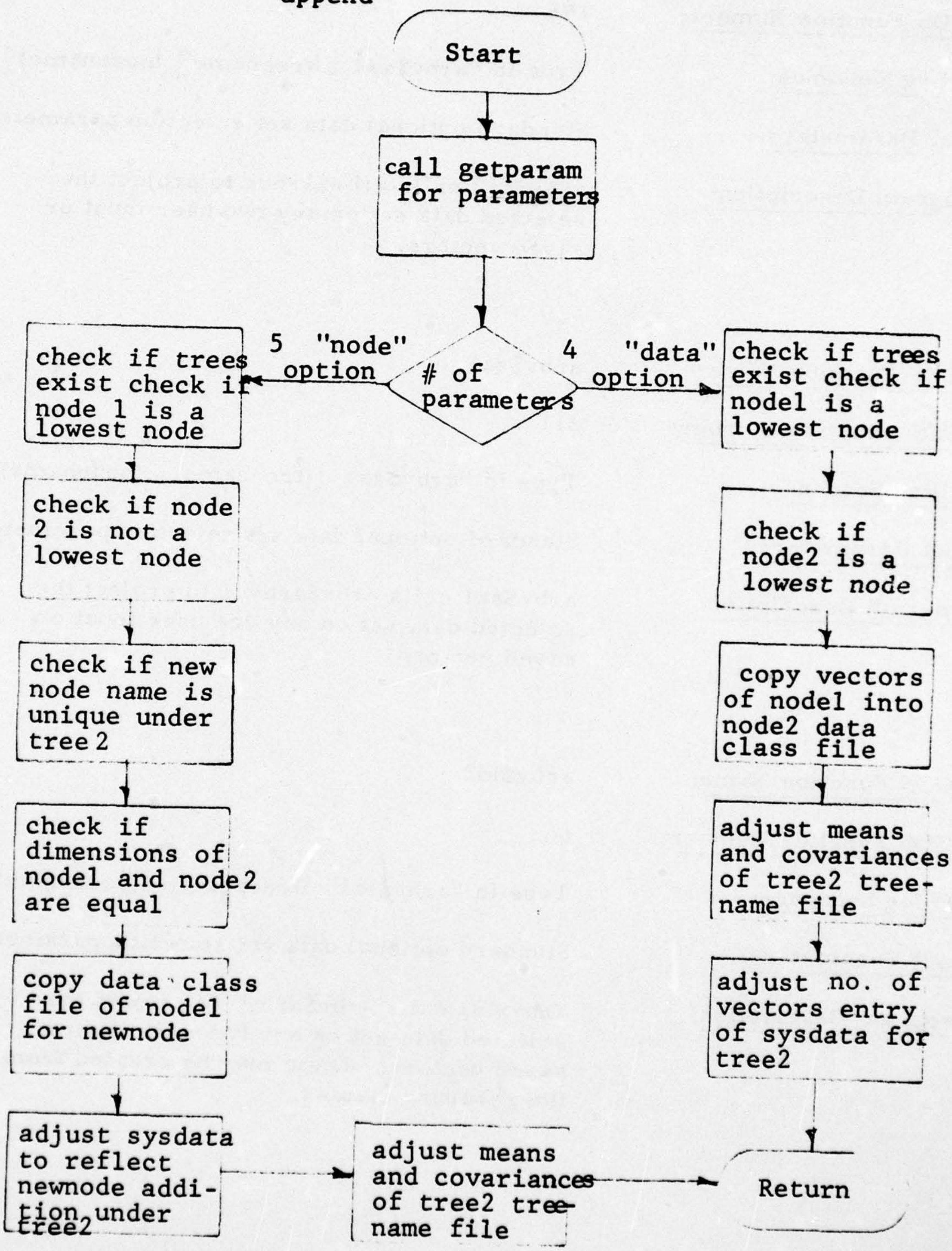
Output File Settings: If a new node is created, then a new data class file corresponding to that node is created and the tree2 treename file is altered. If a new node is not created, then the node2 data class file is adjusted to reflect the addition of nodel data. sysdata file is adjusted to show the changes made in the tree structure.

Program Description: append has two possible options. One, the "node" option, copies a lowest node (nodel) under a tree (treel) as a new lowest node (newnode) under a superior node (node2) of tree2. The "data" option combines a lowest node (nodel) under treel with another lowest node (node2) under tree2.

If the "node" option is selected, then the tree2 (which can also be the same as treel) structure is modified and contains a new lowest node. If the "data" option is selected, the tree2 structure is not modified, but a lowest node in this tree contains its original vectors plus the vectors of another lowest node of a different (or same) tree. append does not modify the first tree, treel, structure.

Flow Chart: See following page.

append



Start
call getparam for parameters
# of parameters
5 "node" option
4 "data" option
check if trees exist check if node 1 is a lowest node
check if node 2 is not a lowest node
check if new node name is unique under tree 2
check if dimensions of nodel and node2 are equal
copy data class file of nodel for newnode
adjust sysdata to reflect newnode addition under tree2
adjust means and covariances of tree2 treename file
check if trees exist check if nodel is a lowest node
check if node2 is a lowest node
copy vectors of nodel into node2 data class file
adjust means and covariances of tree2 treename file
adjust no. of vectors entry of sysdata for tree2
Return

MOOS Function Name: arbv$sa2

MOOS Function Number: 196

Calling Sequence: Type in "arbv$sa2 [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: arbv$sa2 calls arbv$arbvc to project the selected data set on any two user input or saved vectors.

MOOS Function Name: arbv$sal

MOOS Function Number: 211

Calling Sequence: Type in "arbv$sal [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: arbv$sal calls arbv$arbvcl to project the selected data set on any one user input or saved vector.

MOOS Function Name: arbv$ld2

MOOS Function Number: 66

Calling Sequence: Type in "arbv$ld2 [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: arbv$ld2 calls arbv$arbvc to project the selected data set on any two user input or saved vectors. Logic may be created from the resulting display.

MOOS Function Name: arbv$ldl

MOOS Function Number: 81

Calling Sequence: Type in "arbv$ldl [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: arbv$ldl calls arbv$arbvcl to project the selected data set on any one user input or saved vector. Logic may be created from the resulting display.

Internal Subroutine Name: arbv$arbvc
arbv$arbvcl

Calling Sequence:

```
call arbv$arbvc (ptrf, x)
call arbv$arbvcl (ptrf,x)
```

Input Parameters:

ptrf - (5) ptr

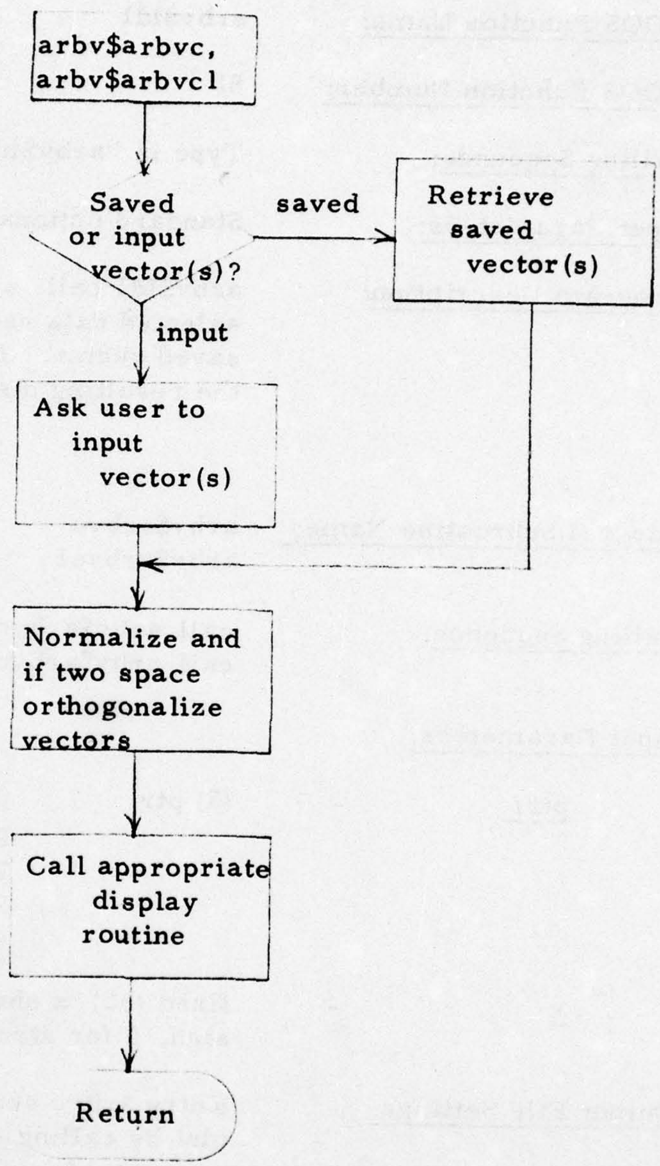
ptrf(1) - sysdata
ptrf(2) - scratch
ptrf(3) - display
ptrf(4) - treename
ptrf(5) - mooslogic

x - fixed (35) x should be set to 1 for logic design, 0 for structure analysis.

Output File Settings: Entry arbvc sets up display for a two-space plot by calling ss$display. Entry arbvcl sets up csdata for a one-space plot by calling ss$displayl.

Program Descriptions: arbvc and arbvcl allows the user to specify saved projection vector(s) or user input projection vector(s). All projection vectors are normalized and orthogonalized. arbvc exits by calling ss$display then clusscat. arbvcl exits by calling ss$displayl then npcos.

Flow Chart: See following page.


arbv$arbvc,
arbv$arbvcl
Saved or input vector(s)?
saved
Retrieve saved vector(s)
input
Ask user to input vector(s)
Normalize and if two space orthogonalize vectors
Call appropriate display routine
Return

MOOS Function Name: ardg$sa2

MOOS Function Number: 199

Calling Sequence: Type in "ardg$sa2 [(treename)][(nodename)]"

Input Parameters: Standard optional data set selection parameters

Program Description: ardg$sa2 calls ardg$arbdg to project the data set on the plane formed by the fisher direction and the orthogonal discriminant direction. These projection vectors are calculated from two user selected groupings of data nodes.

MOOS Function Name: ardg$sa1

MOOS Function Number: 213

Calling Sequence: Type in "ardg$sa1 [(treename)][(nodename)]"

Input Parameters: Standard optional data set selection parameters

Program Description: ardg$sa1 calls ardg$arbdg1 which projects the selected data set on the fisher direction determined by two user selected groupings of data nodes.

MOOS Function Name: ardg$ld2

MOOS Function Number: 68

Calling Sequence: Type in "ardg$ld2 [(treename)] [(nodename)]"

Input Parameters: Standard optional data set selection parameters

Program Description: ardg$ld2 calls ardg$arbdg to project the selected data set on the plane formed by the fisher direction and the orthogonal discriminant direction. These projection vectors are calculated using two user selected groupings of data nodes. Logic may be created from the resulting display.

MOOS Function Name: ardg$ldl

MOOS Function Number: 83

Calling Sequence: Type in "ardg$ldl [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: ardg$ldl calls ardg$arbdgl which projects the selected data set on the fisher direction determined by two user selected groupings of data nodes. Logic may be created from the resulting display.

Internal Subroutine Name: ardg$arbdg
ardg$arbdgl

Calling Sequence: call ardg$arbdg (ptrf, x)
call ardg$arbdgl (ptrf, x)

Input Parameters:

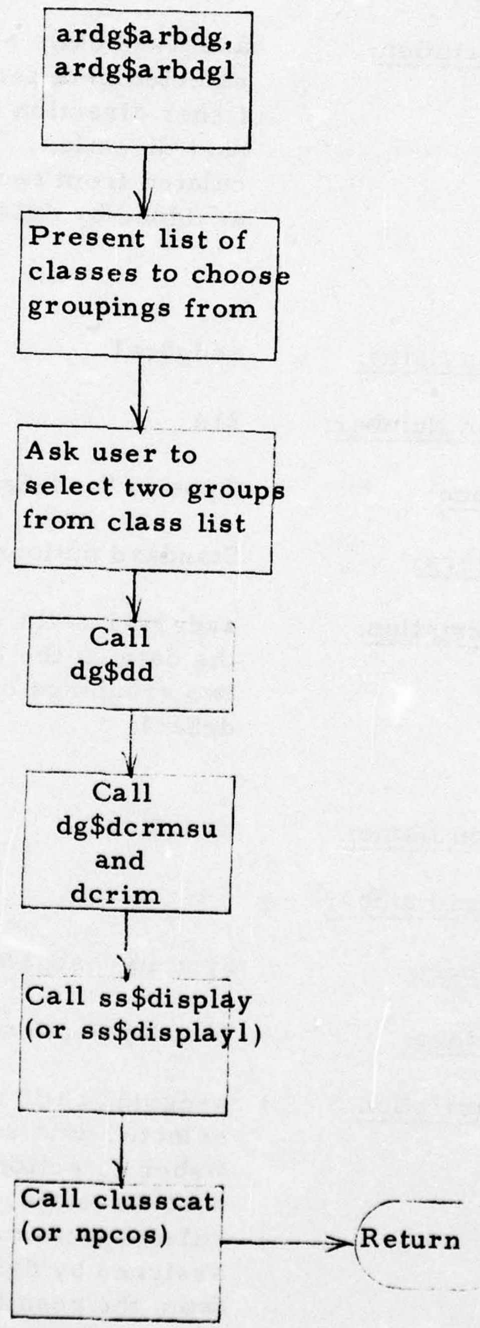
ptrf	(5) ptr	ptrf(1)	-	sysdata
		ptrf(2)	-	scratch
		ptrf(3)	-	display
		ptrf(4)	-	treename
		ptrf(5)	-	mooslogic
x	fixed (35)	x should be set to 1 for logic design, 0 for structure analysis.		

Output File Settings: Entry arbdg sets up display for a two-space plot by calling ss$display. Entry arbdgl sets up csdata for a one-space plot by calling ss$displayl.

Program Description: arbdg and arbdgl present a list of lowest nodes in the selected data set and allows the user to form two groups from this list - not necessarily using all the available classes. The routines then call dg$dd so the user will be given the choice of having the fisher direction based on a scatter matrix or covariance matrix and also the ability to choose which measurements will be included in the fisher direction. dg$dcrmsu and finally dcrim

are called to calculate the fisher direction. The routines exit by calling the appropriate display routine.

Flow Chart:



MOOS Function Name: asdg$sa2

MOOS Function Number: 200

Calling Sequence: Type in "asdg$sa2 [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: asdg$sa2 calls asdg$assdg which projects the selected data set onto the plane formed by the fisher direction and the orthogonal discriminant direction. The fisher direction is calculated from two groupings of lowest nodes assigned by dg$acl.

MOOS Function Name: asdg$sal

MOOS Function Number: 214

Calling Sequence: Type in "asdg$sal [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: asdg$sal calls asdg$assdgl which projects the data on the fisher direction determined by two groupings of lowest nodes, assigned by dg$acl.

MOOS Function Name: asdg$ld2

MOOS Function Number: 69

Calling Sequence: Type in "asdg$ld2 [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: asdg$ld2 calls asdg$assdg which projects the selected data set onto the plane formed by the fisher direction and the orthogonal discriminant direction. The fisher direction is calculated from two groupings of lowest nodes assigned by dg$acl. Logic may be created from the resulting display.

MOOS Function Name: asdg$ldl

MOOS Function Number: 84

Calling Sequence: Type in "asdg$ldl [(treename)] [(nodename)]"

Input Parameters: Standard optional data set selection parameters

Program Description: asdg$ldl calls asdg$assdgl which projects the data on the fisher direction determined by two groupings of lowest nodes, assigned by dg$acl. Logic may be created from the resulting display.

Internal Subroutine Name: asdg$assdg
asdg$assdgl

Calling Sequence:
call asdg$assdg (ptrf, x)
call asdg$assdgl (ptrf, x)

Input Parameters:

ptrf - (5) ptr
ptrf(1) - sysdata
ptrf(2) - scratch
ptrf(3) - display
ptrf(4) - treename
ptrf(5) - mooslogic

x - fixed (35) x should be set to 1 for logic design, 0 for structure analysis.

Output File Settings: Entry assdg sets up display for a two-space plot by calling ss$display. Entry assdgl sets up csdata for a one-space plot by calling ss$display1.

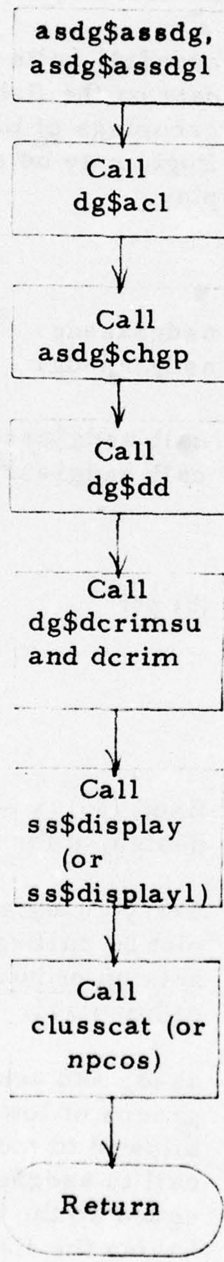
Program Description: assdg and assdgl call dg$acl to assign two groups of lowest nodes. The user is then allowed to modify these groupings through a call to asdg$chgp. The routine then calls dg$dd so the user will be given the choice of having the fisher direction based on a scatter matrix or covariance matrix and also the ability to choose which measurements will be

AD-A033 437 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
MULTICS OLPARS OPERATING SYSTEM.(U)
SEP 76 D B CONNELL, K N KLINGBAIL F30602-75-C-0226
UNCLASSIFIED PAR-74-25-B RADC-TR-76-271-VOL-2 NL

2 OF 7
AD
A033437

included in the fisher direction. dg$dcrmsu and *finally dcrim are called to calculate the* fisher direction. The routines exit by calling the appropriate display routine.

Flow Chart:



MOOS Function Name: asdg$ldl

MOOS Function Number: 84

Calling Sequence: Type in "asdg$ldl [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: asdg$ldl calls asdg$assdgl which projects the data on the fisher direction determined by two groupings of lowest nodes, assigned by dg$acl. Logic may be created from the resulting display.

Internal Subroutine Name: asdg$assdg
asdg$assdgl

Calling Sequence:
call asdg$assdg (ptrf, x)
call asdg$assdgl (ptrf, x)

Input Parameters:

ptrf - (5) ptr

ptrf(1) - sysdata
ptrf(2) - scratch
ptrf(3) - display
ptrf(4) - treename
ptrf(5) - mooslogic

x - fixed (35) x should be set to 1 for logic design, 0 for structure analysis.

Output File Settings: Entry assdg sets up display for a two-space plot by calling ss$display. Entry assdgl sets up csdata for a one-space plot by calling ss$displayl.

Program Description: assdg and assdgl call dg$acl to assign two groups of lowest nodes. The user is then allowed to modify these groupings through a call to asdg$chgp. The routine then calls dg$dd so the user will be given the choice of having the fisher direction based on a scatter matrix or covariance matrix and also the ability to choose which measurements will be

Internal Subroutine Name: asdg$chgp

Calling Sequence: call asdg$chgp (lnd, l, m)

Input Parameters:

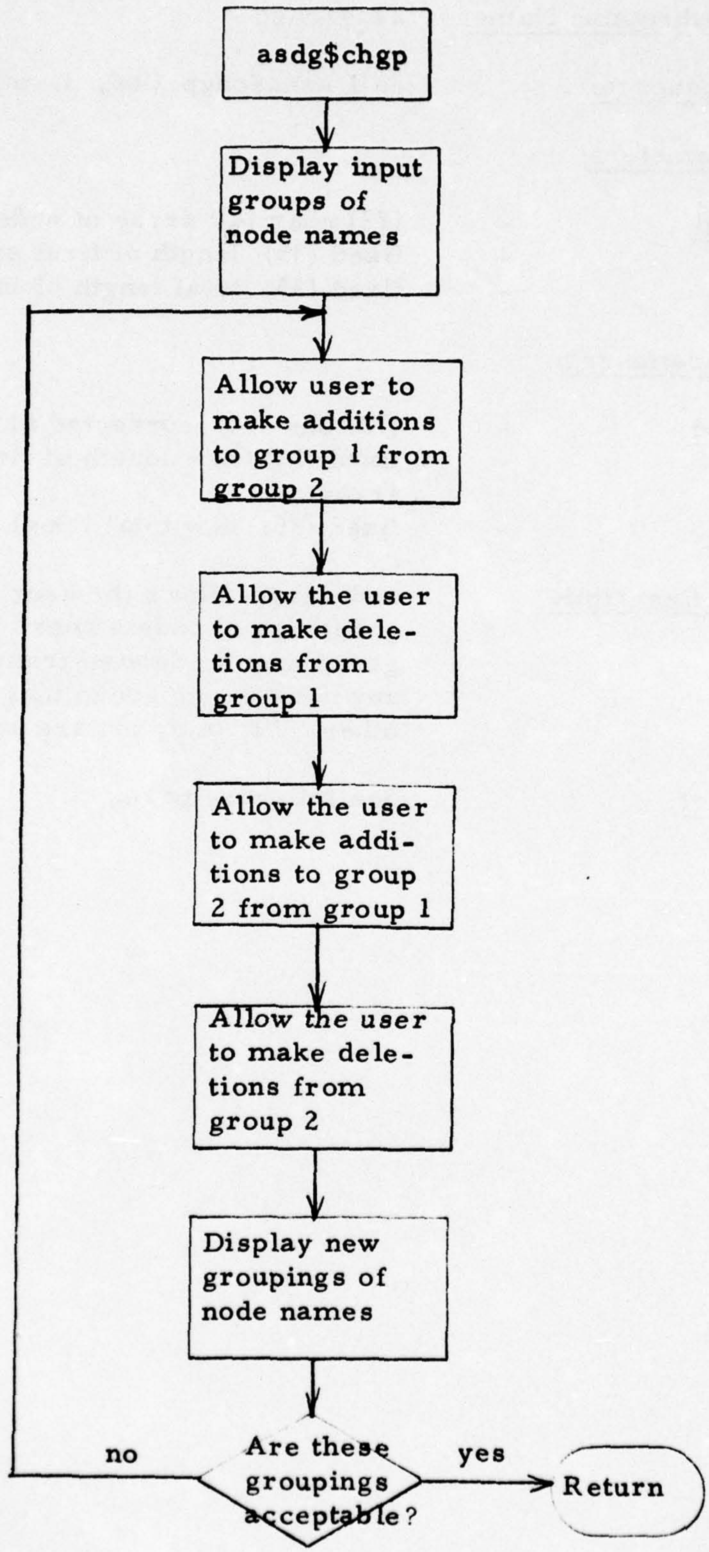
lnd	-	(72) char (4) array of node names.
l	-	fixed (35) length of first group in lnd array.
m	-	fixed (35) total length of lnd array.

Output Parameters:

lnd	-	(72) char (4) corrected array of node names.
l	-	fixed (35) new length of first group in lnd array.
m	-	fixed (35) new total length of lnd array.

Program Description: asdg$chgp allows the user to modify two groupings of node names. Any node in either group may be deleted from that group and any node in one group may be added to the other. "l" and "m" are adjusted accordingly.

Flow Chart: See following page.


asdg$chgp
Display input groups of node names
Allow user to make additions to group 1 from group 2
Allow the user to make deletions from group 1
Allow the user to make additions to group 2 from group 1
Allow the user to make deletions from group 2
Display new groupings of node names
Are these groupings acceptable?
no
yes
Return

Utility Function Name: binwidth

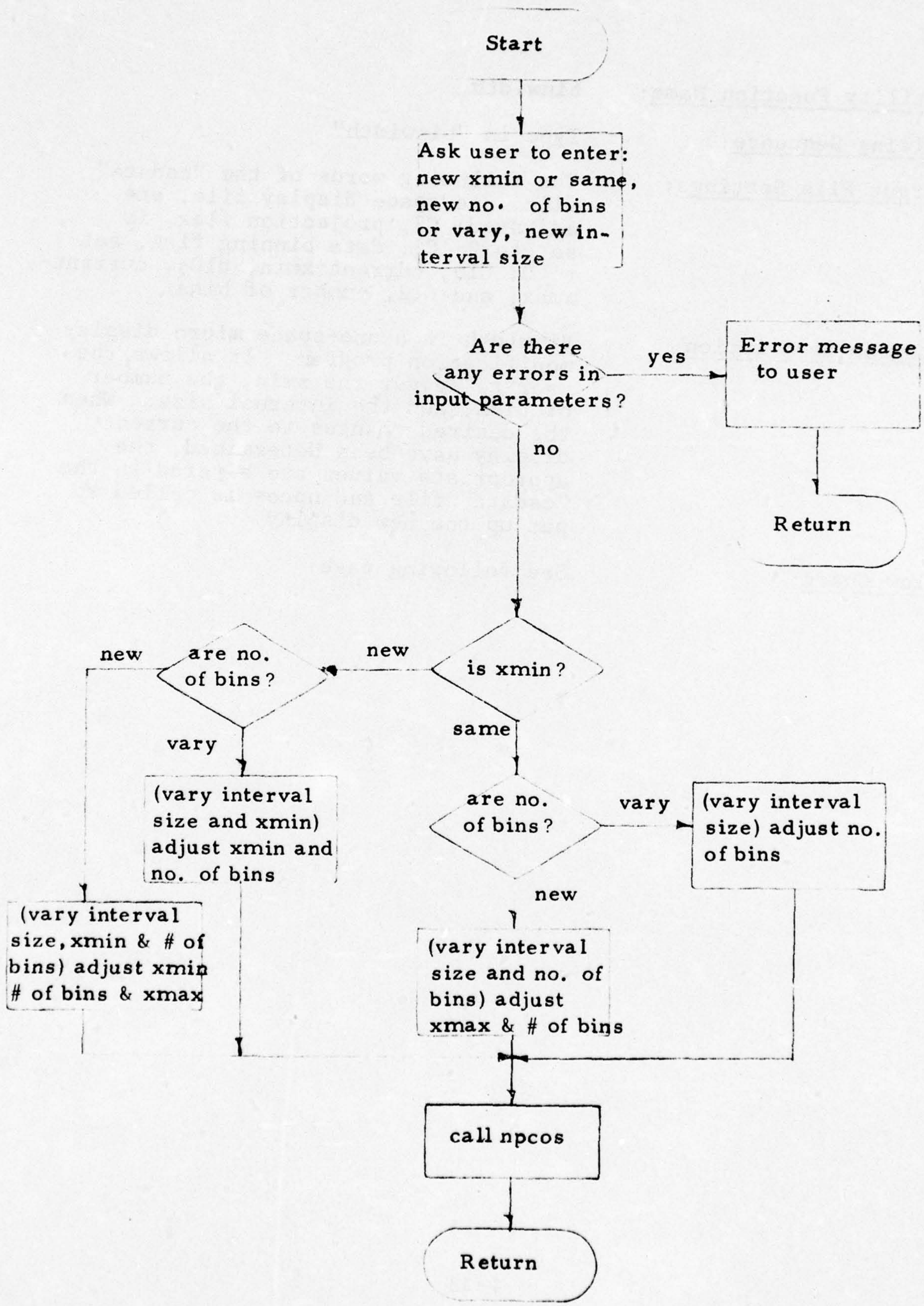
Calling Sequence: Type in "binwidth"

Output File Settings: The following words of the "csdata" file, one-space display file, are adjusted: C7, projection flag, is set to 2; C8, data binning flag, set to 0; C10, current-xmin, $d10_2$, current-xmax; and C12, number of bins.

Program Description: Binwidth is a one-space micro display modification program. It allows the user to adjust the xmin, the number of bins, and the internal size. When the desired changes to the current display have been determined, the appropriate values are altered in the "csdata" file and npcos is called to put up the new display.

Flow Chart: See following page.

binwidth


Start
Ask user to enter: new xmin or same, new no. of bins or vary, new interval size
Are there any errors in input parameters?
yes
Error message to user
Return
no
is xmin?
new
are no. of bins?
new
vary
(vary interval size and xmin) adjust xmin and no. of bins
(vary interval size, xmin & # of bins) adjust xmin # of bins & xmax
same
are no. of bins?
vary
(vary interval size) adjust no. of bins
new
(vary interval size and no. of bins) adjust xmax & # of bins
call npcos
Return

Internal Subroutine Name: boologic

Calling Sequence: **call boologic (ptr, ndim, boolval, num)**

Input Parameters:

ptr	-	a pointer to the vector to be evaluated
ndim	-	dimensionability of the vector
num	-	logic node number of the statement to be evaluated

Output Parameters:

boolval	-	a 1 bit variable and it tells whether evaluation of the vector was true or false

Program Description: "boologic" is a routine generated by other routines (logen, etc.) Its purpose is to evaluate a vector against a given set of constraints, (these constraints are not known until the routine that generated "boologic" is run). Therefore no flow chart can be written for "boologic."

Internal Subroutine Name: box_logic

Calling Sequence: call box_logic (lptr, nptr, eptr, dptr, ndim, nnum, trcl, cn)

Input Parameters:

lptr	ptr pointer to logic file
nptr	ptr pointer to closed decision boundary logic block
eptr	ptr pointer to error entry in box_error_file
dptr	ptr pointer to OLPARS vector to be evaluated
ndim	fixed (35) no. of dimensions
nnum	(128) fixed (35) array of indices, where the argument of nnum = a logic node number, and the value of nnum = an index to the class name associated with the logic node number. (An index of 1 would refer to the first SCl entry in the logic file).
trcl	fixed (35) logic node number of the correct class.

Output Parameters:

cn	fixed (35) assigned logic node number.

Program Description:

box_logic determines into which closed decision boundary(ies) a single OLPARS vector falls. The routine tests the vector against the boundary surrounding each class, and if the vector lies inside, a counter is bumped and the specific boundary noted. Returned information consists of the parameter cn, and error information (if the vector is not correctly classified) in the box_error_file.

For a more detailed description of the operation of box_logic, see the program listing documentation.

Internal Subroutine Name: boxprogram

Calling Sequence: call boxprogram (ii, logicptr)

Input Parameters:

ii — logic node number with closed decision logic (fixed (35))

logicptr — pointer to MOOS logic file (ptr)

Program Description: boxprogram is the subroutine in the "fortlogc" option of MOOS which converts closed decision logic into its FORTRAN equivalent

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: cdblogic

Calling Sequence: call cdblogic (lptr, ndim, block, class, d)

Input Parameters:

lptr	ptr pointer to a logic file
ndim	fixed (35) no. of dimensions
block	fixed (35) index to a closed decision boundary logic block
class	(72) char (4) array of class names associated with the closed decision boundary logic node.
d	char (12) name of the output file (listlog_file)

Program Description:

cdblogic is utilized by listlogc to print out the numerical values which specify closed decision boundary logic.

For a more detailed description of the operation of cdblogic, see the program listing documentation.

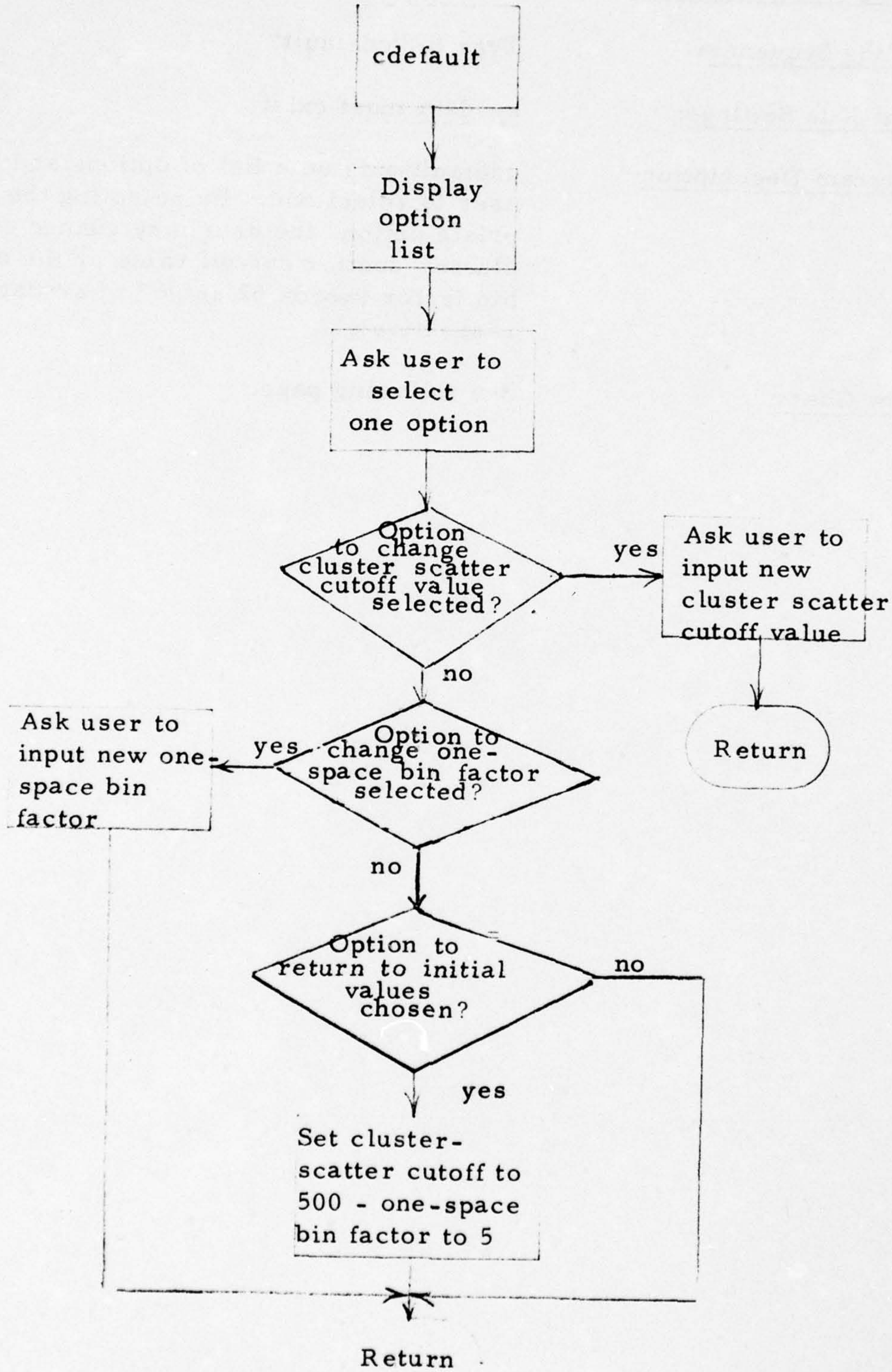
Utility Function Name: cdefault

Calling Sequence: Type in "cdefault"

Input File Settings: sysdata must exist

Program Description: cdefault puts up a list of options and asks the user to select one. By selecting the appropriate option, the user may change either the *cluster scatter* cut-off value or the one-space bin factor (words 62 and 63 of sysdata, respectively).

Flow Chart: See following page.


cdefault
Display option list
Ask user to select one option
Option to change cluster scatter cutoff value selected?
yes
Ask user to input new cluster scatter cutoff value
Return
no
Ask user to input new one-space bin factor
yes
Option to change one-space bin factor selected?
no
Option to return to initial values chosen?
no
yes
Set cluster-scatter cutoff to 500 - one-space bin factor to 5
Return

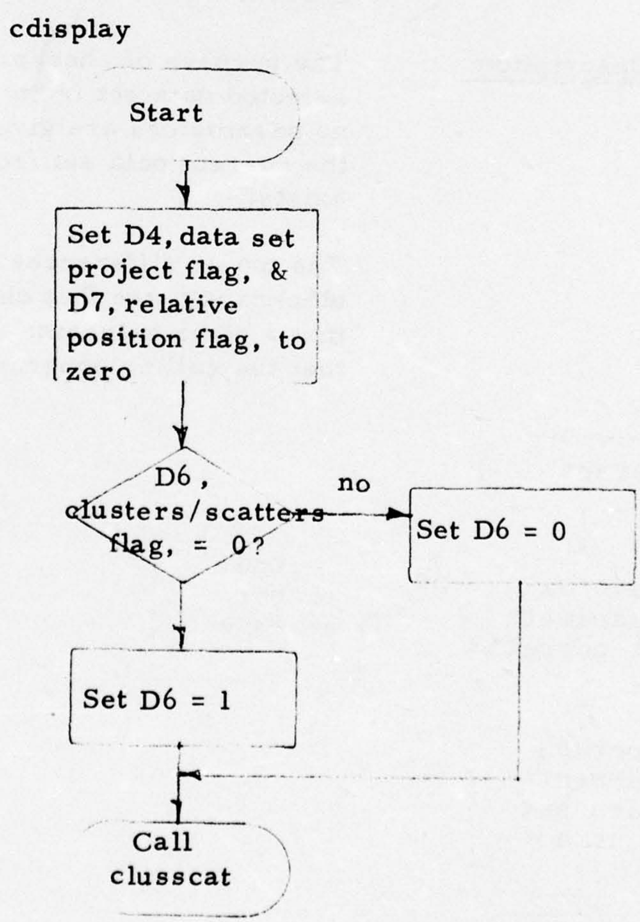
Utility Function Name: cdisplay

Calling Sequence: type in "cdisplay"

Output File Settings: The "display" file is adjusted to reflect the changed two-space display.

Program Description: This program changes the current display from a cluster plot to a scatter plot. cdisplay checks word D6, the cluster/scatter flag, and reverses the value. The program exits by calling clusscat.

Flow Chart:



Internal Subroutine Name: checkp

Calling Sequence: call checkp (trnam, cnam, m, sptr).

Input Parameters:

sptr - ptr pointer to parameter list (as returned by cu_$arg_list_ptr).

Output Parameters:

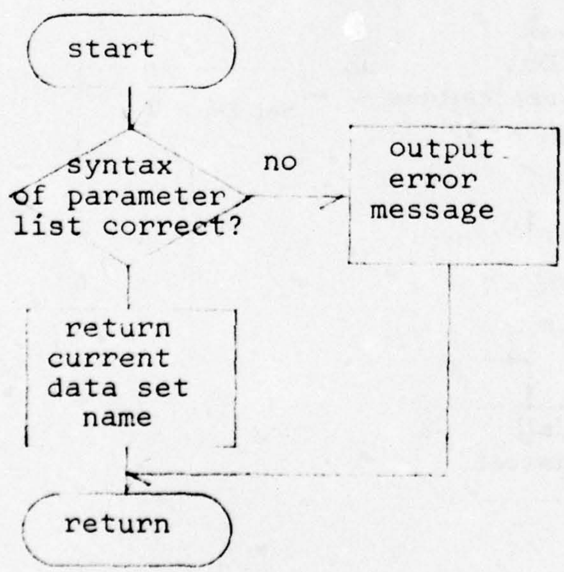
trnam - char (8) treename of current/or selected data set.

cnam - char (4) nodename of current/or selected data set.

m - fixed (35) set to -1 if checkp finds any errors.

Program Description: The purpose of checkp is to return the selected data set from a parameter list. If no parameters are given, checkp will return the current data set from sysdata, if one exists.

The major differences between checkp and ut$ckparam are that checkp does not require that a given treename exist in sysdata or that the calling program be a MOOS function.



MOOS Function Name: chngaprb

MOOS Function Number: 30

Calling Sequence: Type in "chngaprb [(treename)] [(nodename)] "

Input Parameters: The standard optional data set selection parameters

Output Parameters: "treename nodename logic" file: The apriori probabilities in the logic file will be updated according to user input information.

Program Description: chngaprb calls chngaprb$display to list the apriori probabilities and to allow the user to input his option, and alter them according to user input option:

0 - no change
1 - set all apriori probabilities $P(i) = 1 \div (ncls)$
2 - enter proportion for each class

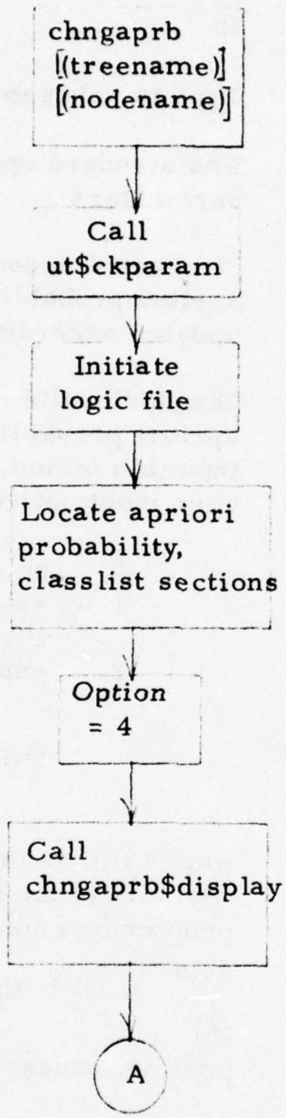
$$P(i) = 1 \div \sum_{j=1}^{ncls} \left[wt(j)\right] \times wt(i)$$

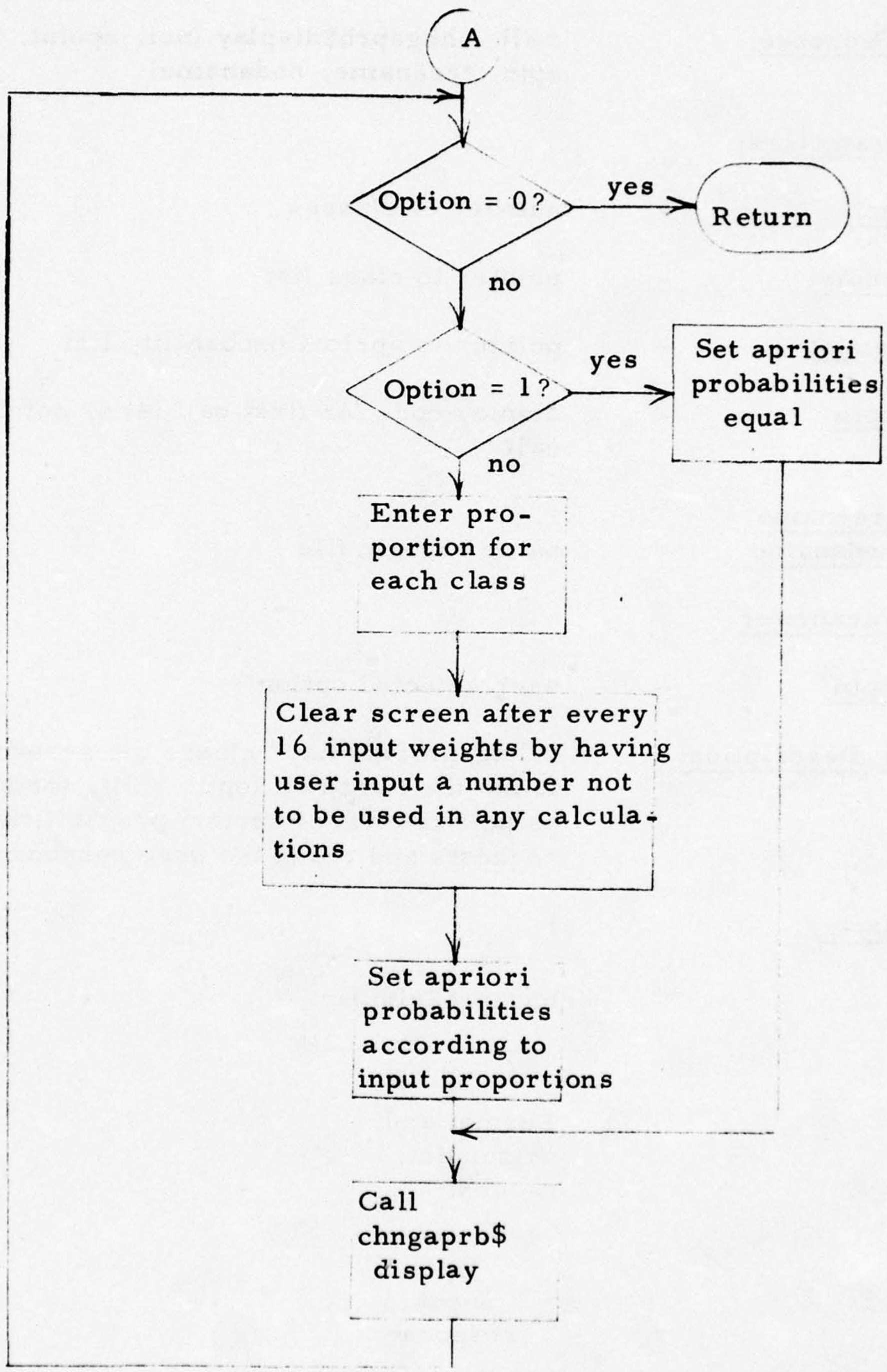
where P(i) is the apriori probability for class (i), ncls is the number of classes, wt(i) is the proportion entered for class(i), and $\sum_{j=1}^{ncls}$ wt(j) is the sum of all user input proportion values.

chngaprb$display is called after each change to list the apriori probabilities and to allow the user to enter his option.

The routine exits normally when the user selects option 0, and exits with an error message if ut$ckparam indicates an error or if no logic file is found for (treename), (nodename).

Flow Chart: See following page.


chngaprb
[(treename)]
[(nodename)]
Call
ut$ckparam
Initiate
logic file
Locate apriori
probability,
classlist sections
Option
= 4
Call
chngaprb$display
A


A
Option = 0?
yes
Return
no
Option = 1?
yes
Set apriori probabilities equal
no
Enter proportion for each class
Clear screen after every 16 input weights by having user input a number not to be used in any calculations
Set apriori probabilities according to input proportions
Call chngaprb$ display

Internal Subroutine Name: chngaprb$display

Calling Sequence: call chngaprb$display (ncl, cpoint, ppoint, optn, treename, nodename)

Input Parameters:

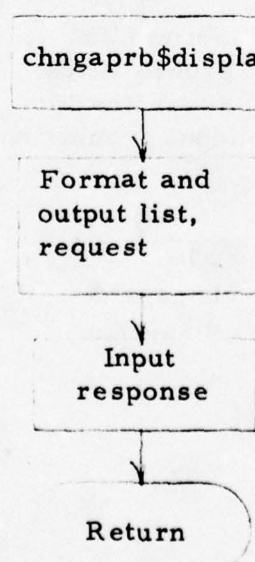
- ncl - number of classes
- cpoint - pointer to class list
- ppoint - pointer to apriori probability list
- optn - display code for first call (4) or not first call
- treename, nodename - name of logic file

Output Parameter

- optn - user selected option

Program Description: "chngaprb$display" clears the screen if it is not the first call (optn $\neq$ 4), formats and outputs the list of apriori probabilities, and requests and reads the user response.

Flow Chart:



MOOS Function Name: chngname

MOOS Function Number: 29

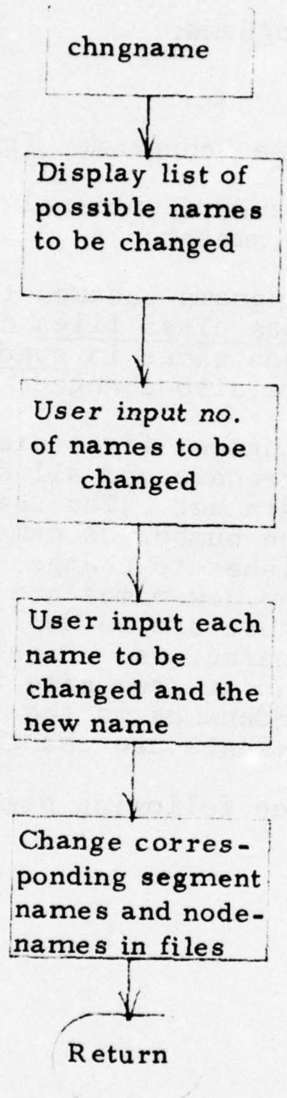
Calling Sequence: Type "chngname [(treename) (nodename)]

Input Parameters: Standard optional data set selection parameters

Output File Settings: chngname renames the treename file and/or data class files of the selected data set. Node names in sysdata and treename files are also changed.

Program Description: chngname first displays the selected treename and all nodes in the selected data set. The user is asked to enter the number of names from this list he wishes to change - followed by the changes. The new names are checked to insure that there are no duplicated "display" characters. Finally, the data class files and/or tree name file are renamed and nodenames in the tree name file and sysdata are changed.

Flow Chart: See following page.


chngname
Display list of possible names to be changed
User input no. of names to be changed
User input each name to be changed and the new name
Change corresponding segment names and node-names in files
Return

Internal Subroutine Name: cl_restruct

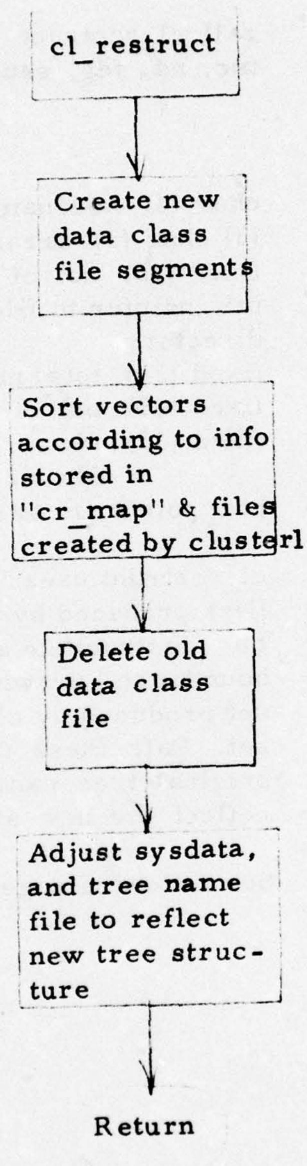
Calling Sequence: call cl_restruct (cnam, llow, nln, cptr, tnc, nd, fdg, ssptr)

Input Parameters:

cnam	-	char (4) nodename of selected data set.
llow	-	(3) char (4) array of new dataclass names.
nln	-	fixed (35) no. of new dataclasses.
cptr	-	ptr pointer to file cr_map in process directory.
tnc	-	fixed (35) total number of cluster centers.
nd	-	fixed (35) no. of dimensions.
fdg	-	fixed (35) fdg = 1 if 2 boundaries fdg = 2 if 1 boundary.
ssptr	-	ptr pointer to sysdata.

Program Description: cl_restruct uses "cr_map" and the mapping files produced by internal subroutine cluster1 to restructure an original data set based on boundary(s) drawn on a projection of a data set produced by clustering the original data set. Data class files, sysdata, and the original tree name file are adjusted to reflect the new structure.

Flow Chart: See following page.


cl_restruct
Create new data class file segments
Sort vectors according to info stored in "cr_map" & files created by clusterl
Delete old data class file
Adjust sysdata, and tree name file to reflect new tree structure
Return

Utility Function Name:	cleartree
Calling Sequence:	type in "cleartree (treename/"all")"
Input Parameters:	
treename	specify a particular data set
"all"	perform operation for all data sets in the user's "saved_trees" directory.
Output File Settings:	
Directory -	user's "saved_trees" will be reduced.
Tables -	user's "seg_o_trees" will be reduced.
Segment -	user's "structure" will be reduced.
Program Description:	This routine calls s_p$tclr and returns control to the user. The files "seg_o_trees" and "structure" will show the reduction in data sets.

MOOS Function Name: closedcn

MOOS Function Number: 98

Calling Sequence: Type in "closedcn ((treename)) ((classname))"

Input Parameters: Standard optional data set selection parameters.

Output File Settings: A closed decision boundary logic block is added to the logic file.

Program Description:

closedcn creates closed decision boundary logic for the selected data set at a user-specified logic node. closedcn is composed of two major sections. Section I consists of interactive user input of the logic specification for each class. Section II creates the desired closed decision boundary logic. The routine ends by calling pevbx which produces a partial evaluation of closed decision boundary logic.

For a more detailed description of the operation of closedcn, see the program listing documentation.

MOOS Function Name: closemod

MOOS Function Number: 99

Calling Sequence: Type in "closemod ((treename)) ((classname))"

Input Parameters: Standard optional data set selection parameters.

Program Description:

closemod modifies closed decision boundary logic at a user-specified logic node in a selected logic tree. closemod operates in the following manner: the class whose logic is to be modified and the type of modification are first specified by the user. closemod then performs the desired changes and the user is allowed to choose another class to modify or stop. When all modification is complete, a partial evaluation of the new logic is generated by pevbx. An internal subroutine "closemod$thres" is utilized in the calculation of thresholds for hyperrectangular logic.

For a more detailed description of the operation of closemod, see the program listing documentation.

Internal Subroutine Name: close_ut

Calling Sequence: call close_ut (lptr, node, opt1, opt2)

Input Parameters:

lptr — ptr pointer to a logic file

node — fixed (35) current logic node

Output Parameters:

node — fixed (35) node no. of class to be modified.

opt1 — char (1) 1st option list

opt2 — char (8) 2nd option list

Program Description:

close_ut performs two functions for program closemod. The first function is to allow the user to select one of the classes at the current logic node for modification. The second function is to allow the user to select a logic type for the selected class. The logic type is returned as the 1st option list (opt1). The second option list (opt2) consists of the modification options chosen by the user for the selected logic type.

For a more detailed description of the operation of close_ut, see the program listing documentation.

Internal Subroutine Name: close_ut$bndy

Calling Sequence: call close_ut$bndy (type)

Input Parameters:

type — fixed (35) 1 for hyperrectangular, 2 for hyperellipsoid

ptrs — (5) ptr external static

- ptrs (1) - sysdata
- ptrs (2) - scratch
- ptrs (3) - display
- ptrs (4) - treename
- ptrs (5) - mooslogic

low — float external static current low threshold

high — float external static current high threshold

thptr — ptr external static pointer to thresholds in the logic file

Program Description:

close_ut$bndy allows the user to change thresholds interactively for hyperrectangular and hyperellipsoid closed decision boundary logics. For hyperellipsoid logic, a threshold change is the same as changing the axis length along one of the axis vectors.

For a more detailed description of the operation of close_ut$bndy, see the program listing documentation.

Internal Subroutine Name: close_ut$cc

Calling Sequence: call close_ut$cc(fl)

Input Parameters:

ptrs (5) ptr external static

ptrs(1) - sysdata
ptrs(2) - scratch
ptrs(3) - display
ptrs(4) - treename
ptrs(5) - mooslogic

Output Parameters:

fl fixed (35) set to -1 for error condition

Program Description:

close_ut$cc performs the following tasks for closemod. The logic file for the current data set is initiated. If it does not exist, fl is set to -1 and the routine exits.

If the logic file exists, it is searched for closed decision boundary logic nodes. If there is only one, the routine makes this node the current logic node and returns. If there are more than one, a list is presented and the user may choose a logic node to be the current logic node.

For a more detailed description of the operation of close_ut$cc, see the program listing documentation.

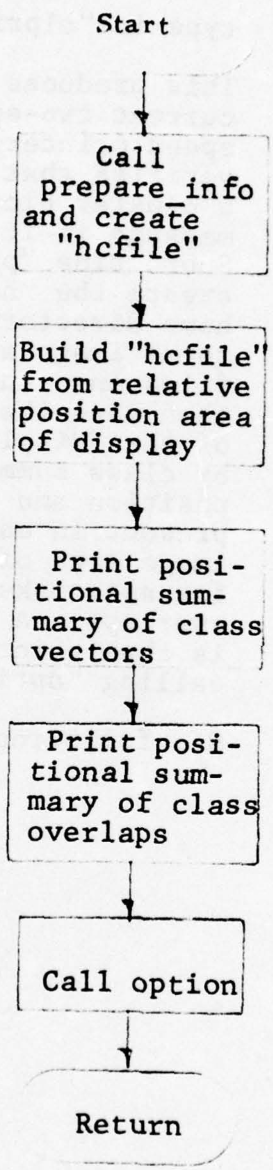
Utility Function Name: clprint

Calling Sequence: type in "clprint"

Program Description: This produces a hardcopy of the current two-space display on a high-speed printer. Initially, clprint verifies that the current display is a cluster plot and prints an error message if it is a scatter plot. Subroutine "prepare_info" is called to create the "hcfile" in the user's home directory. Clprint copies the total image area from the "display" files into the "hcfile." Clprint then uses the relative position area of the "display" to print a class by class summary of row and column position and the number of vectors present in each grid. The total image area of "display" is scanned for asterisks, indicative of class overlaps. A table of class overlaps is then printed. Clprint exits by calling "option".

Flow Chart: See following page.

clprint


Start
Call prepare_info and create "hcfile"
Build "hcfile" from relative position area of display
Print positional summary of class vectors
Print positional summary of class overlaps
Call option
Return

Internal Subroutine Name: clusscat

Calling Sequence: call clusscat

Input File Settings: The following locations of "display" file, two-space format, must be set prior to calling clusscat.

D0	-	system display code, set to 0
$D0_A$	-	temporary symbol set to appropriate class symbol or 0
D1	-	tree character
D2	-	dimensionability
D3	-	number of classes
D4	-	data set projection flag
$D5_1$-$D5_4$	-	"original" data range set depending on value of D4
D6	-	cluster/scatter plot flag
D7	-	relative position flag
$D8_1$-$D8_4$	-	"current" data range, set depending on value of D4
$D9_1$	-	sequence flag
$D9_2$	-	sequence number
D10	-	cycle option
D11	-	cycle data, set according to value of D10
$D12_{1,1}$-$D12_{1,ncls}$	-	classnames
$D12_{2,1}$-$D12_{2,ncls}$	-	eliminate flag
$D12_{3,1}$-$D12_{3,ncls}$	-	intensify flag
$D13_1$	-	number of boundaries
$D13_2$	-	redraw-boundary flag
$D13_3$	-	number of points in boundary 1
$D13_4$	-	number of points in boundary 2
$D14_1$-$D14_{20}$	-	x,y coordinates of boundary pts and convex pts of boundaries 1 and 2
$D15_1$-$D15_{2*ndim}$	-	x,y projection vectors

Output File Settings: The "csdata" and "display" files are adjusted to reflect the current two-space display.

Program Description: The program is the two-space display routine. clusscat first checks if the system display code is 0 or 1, 0 if this is the first time clusscat is entered and 1 if the current display is a two-space plot. If the code is correct, the input file settings are assigned to local variables, else the error message "system display code is incorrect" is printed and control is returned to the calling program. The D4 flag is evaluated next. If D4 = 0 or 1, the data is projected upon the basis vectors and stored in the "csdata" file. If D4 equals 0, then the xmin, xmax, ymin, and ymax values are determined then adjusted so that the x range and y range are equal. These values are then stored as the "original" data range. The D6 flag, cluster/scatter plot, is checked. If a cluster plot is desired then the D7 flag, relative-position, is evaluated and if D7 equals 0 the relative position of each vector, which is the location on the 60x36 cluster grid where the vector lies, is:

$$relpos = yloc*60 + xloc$$

where

$$yloc = \frac{(ymax-y)*36}{(ymax-ymin)}$$

$$xloc = \frac{(x-xmin)*60 + 1}{(xmax-xmin)}$$

x, y are the projected values of the vector

xmin, xmax, ymin, and ymax is the "current data range

The relative position area for each class is constructed and contains each vector's relative position and the number of vectors of each class which fall into this grid.

The total image area, a 60 x 36 word section of the "display" file, is then constructed and is the plot which appears on the console. Each word which represents one grid of the cluster plot, contains either a class symbol or an asterisk if more than one class was projected into the grid. The total image area is the output to the console.

If D6 indicates a scatter plot, each vector of each class that is to be displayed is checked to see if it is within the current data range. If so, its location on the console is determined as follows:

$$\text{xloc} = \frac{(\text{x}-\text{xmin})*714}{\text{xmax}-\text{xmin}} + 120$$

$$\text{yloc} = \frac{(\text{y}-\text{ymin})*670}{\text{ymax}-\text{ymin}} + 53$$

where x, y, xmin, ymin, xmax, and ymax are the same as in the cluster plot.

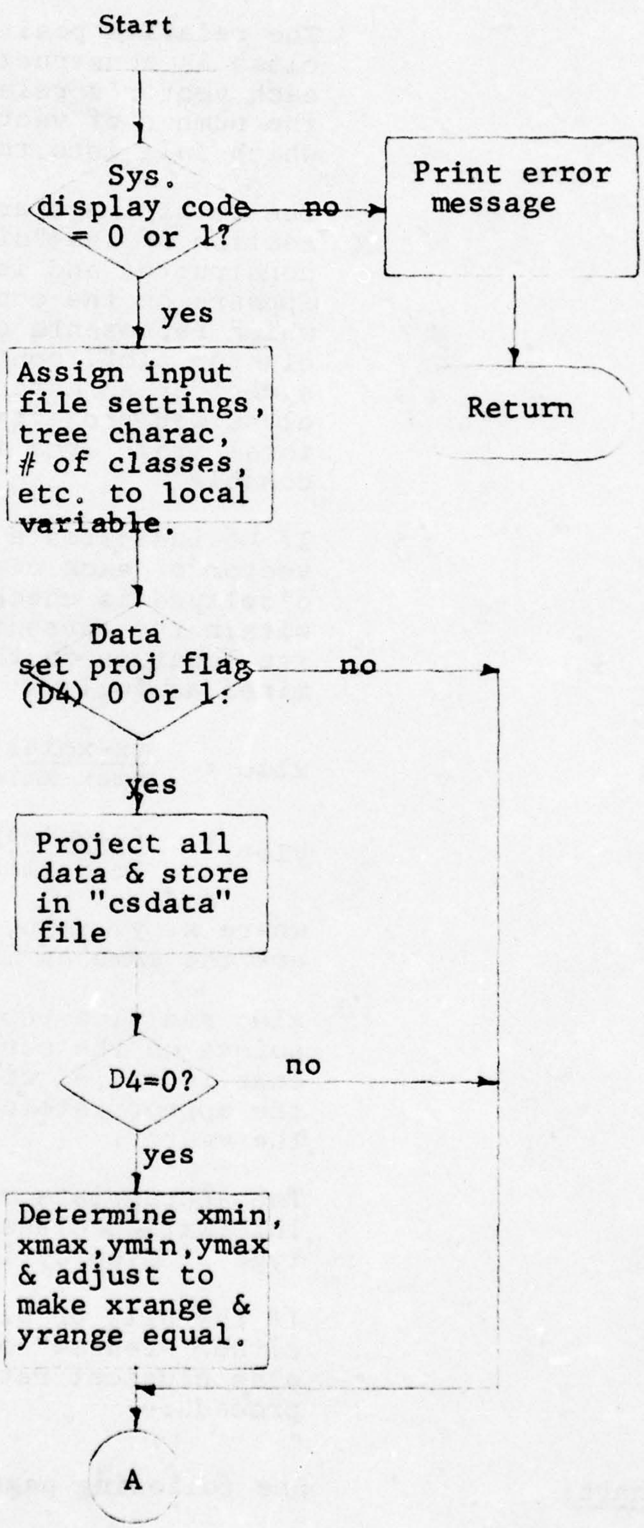
xloc and yloc represent tektronix points on the screen. multeks$ printchar is called with these values and the appropriate class symbol to display the vector.

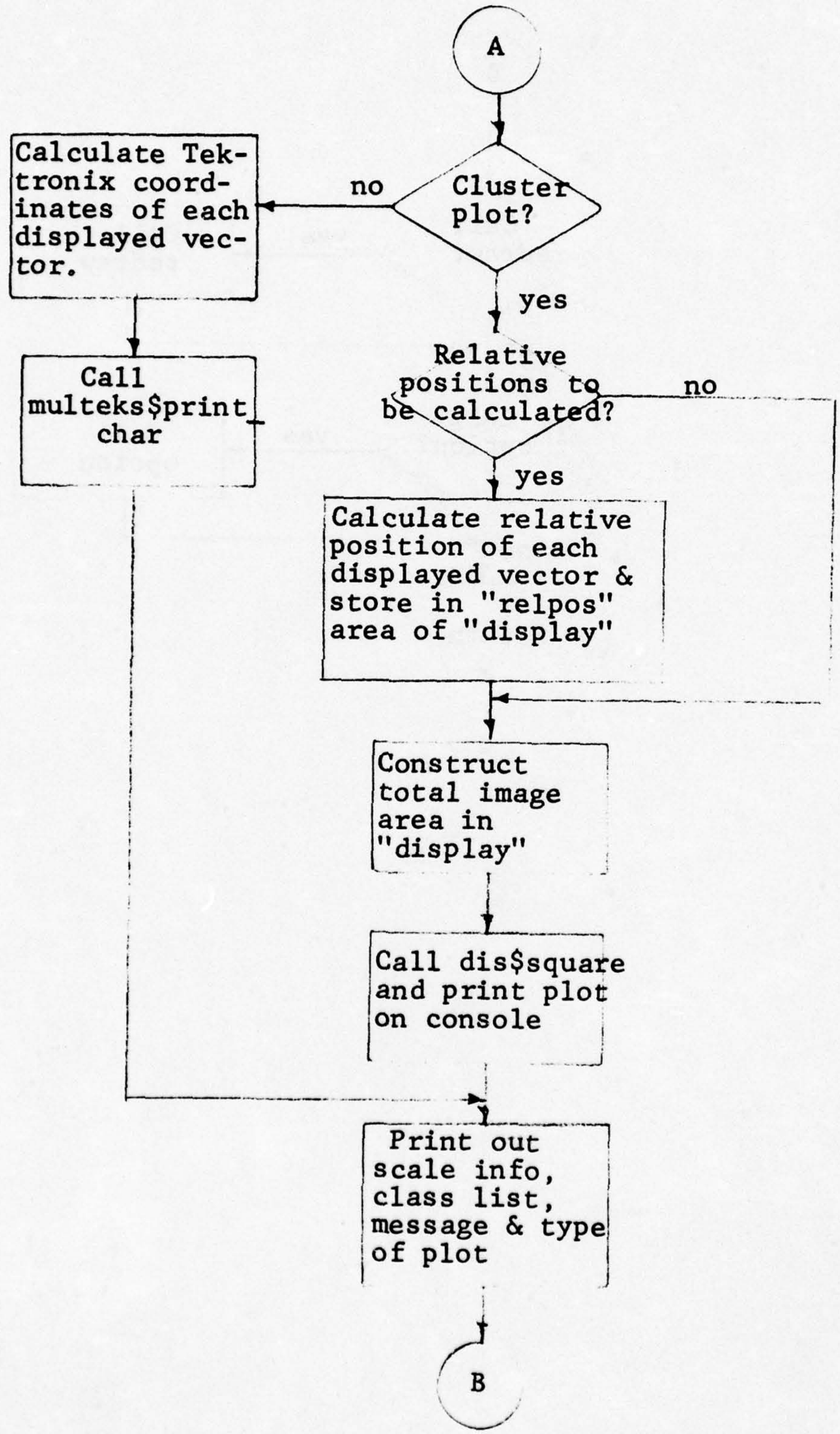
The appropriate range and scaling information, class list, message, and type of display is then printed.

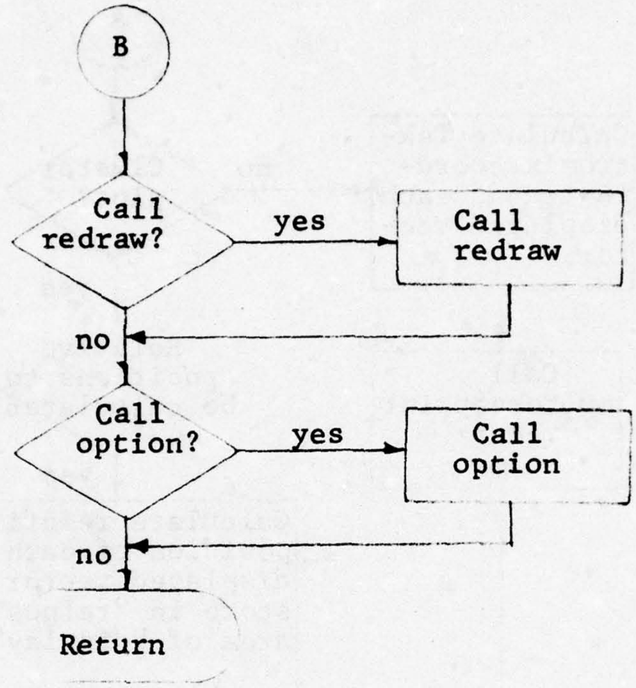
If the $D11_A$ or $D13_2$ switches are set, either "redraw" or "option" is called, else clusscat returns to the calling procedure.

Flow Chart: See following page

clusscat


Start
Sys. display code = 0 or 1?
no
Print error message
Return
yes
Assign input file settings, tree charac, # of classes, etc. to local variable.
Data set proj flag (D4) 0 or 1?
no
yes
Project all data & store in "csdata" file
D4=0?
no
yes
Determine xmin, xmax,ymin,ymax & adjust to make xrange & yrange equal.
A


A
Cluster plot?
no
Calculate Tektronix coordinates of each displayed vector.
Call multeks$print_char
yes
Relative positions to be calculated?
no
yes
Calculate relative position of each displayed vector & store in "relpos" area of "display"
Construct total image area in "display"
Call dis$square and print plot on console
Print out scale info, class list, message & type of plot
B


B
Call redraw?
yes
Call redraw
no
Call option?
yes
Call option
no
Return

Internal Subroutine Name: clusterl

Calling Sequence: call clusterl (ln, nln, sptr, totvec, deptr, name, node)

Input Parameters:

ln	(72) char(4) array of lowest node names in the data set
nln	fixed (35) no. of lowest nodes
sptr	(5) ptr sptr(1) - sysdata sptr(2) - scratch sptr(3) - display sptr(4) - treename sptr(5) - mooslogic
totvec	fixed (35) total number of vectors in the data set
deptr	(72)ptr array of pointers pointing to data class files in the data set
name	char(8) treename of data set to be clustered
node	char(4) nodename of data set to be clustered

Input File Settings: Word 6 of sysdata must be set to 1, and words 7 and 8 to the treename of the data set being clustered. These words are used by restruct and cl_restruct in case the tree produced by clusterl is restructured.

Output File Settings: Clusterl produces a mapping file for each data class. These files contain lists of vector I.D. numbers from the original data classes for each cluster center. If the clustered data set is restructured, cl_restruct uses these files to map back to the original data tree. The mapping files are named as follows: clust_ || treename || nodename. Clusterl also sets up filedata for file_input.

Program Description: Clusterl first asks the user to enter a number of cluster centers. The data tree produced will contain this number of vectors and will approximately retain the structure of the original data set. The user is then asked how the number of representative

clusterl (cont'd)

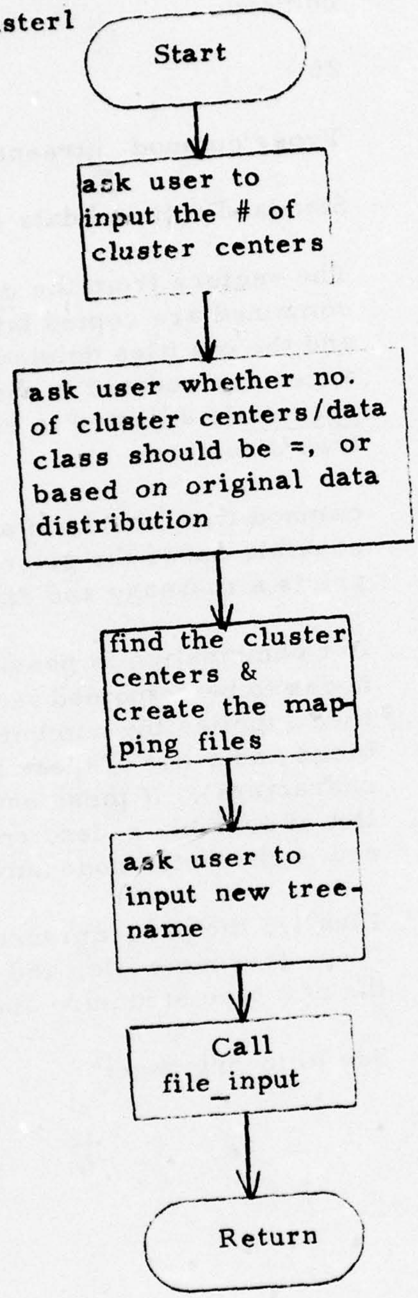
cluster centers is to be distributed among the data classes. Each data class can be represented by an equal number of cluster centers, or the number may be based on the original data distribution.

The clustering technique proceeds in the following manner: the vector whose Euclidean distance from the mean is greatest is the seed of the first cluster. If there are to be n vectors per cluster, the n-1 closest vectors to the first seed are found. The mean of these n vectors forms the first cluster center and the first vector in the reduced data tree. The method for finding succeeding "seed" vectors is to find the vector farthest from the current seed vector simultaneous with finding the n-1 closest vectors.

When all the cluster centers have been located, the routine asks for a new treename for the clustered tree and calls file_input to create the new tree.

Flow Chart: See following page.

cluster1


Start
ask user to input the # of cluster centers
ask user whether no. of cluster centers/data class should be =, or based on original data distribution
find the cluster centers & create the mapping files
ask user to input new tree-name
Call file_input
Return

MOOS Function Name: comnod

MOOS Function Number: 26

Calling Sequence: Type "comnod [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

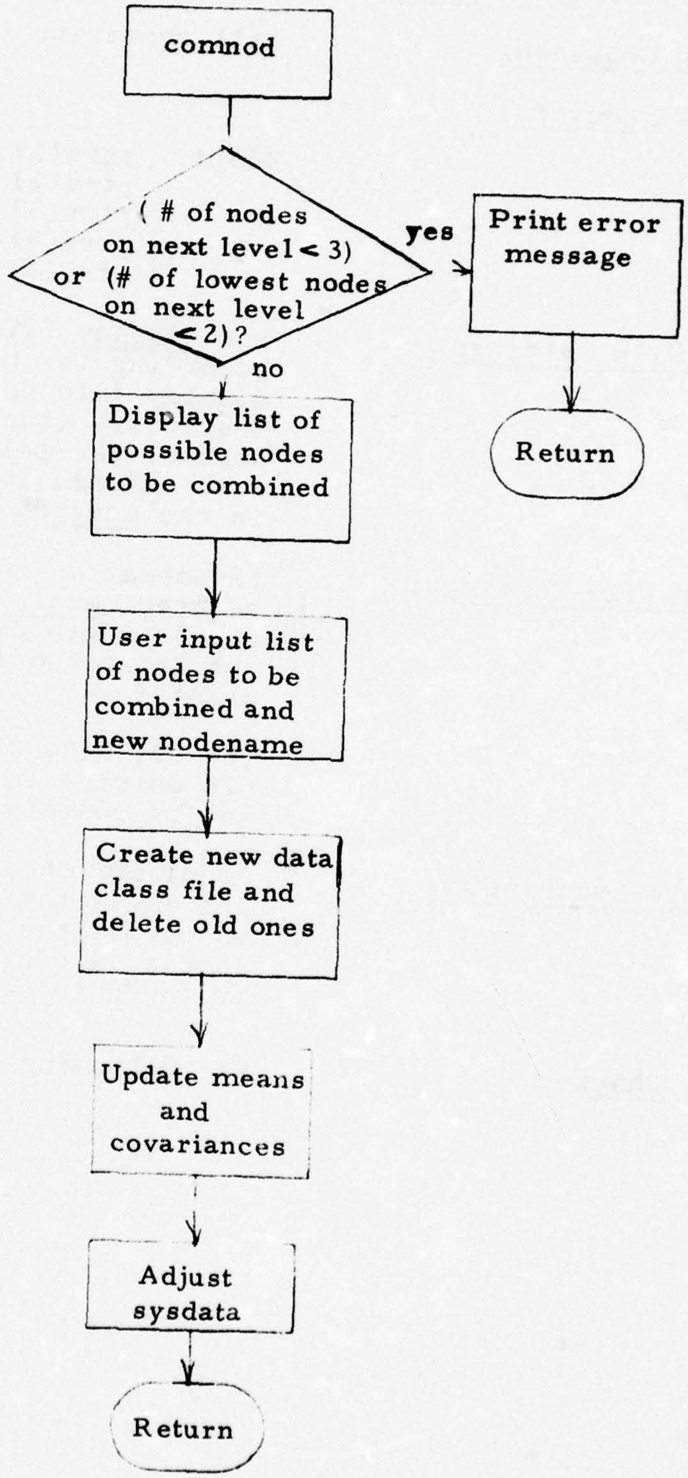
Output File Settings: The vectors from the data class files to be combined are copied into a new data class file and the old files deleted. Mean and covariances are updated in the treename file, and sysdata is adjusted to reflect the modified tree structure.

Program Description: comnod first checks if a combination is possible under the given node - if not, it prints a message and exits.

If a combination is possible, a list of possible nodes to be combined is presented. The user then supplies the number of nodes to be combined and a list of these nodes (display characters). If these entries agree with the list of possible nodes, comnod asks for a new 4 character nodename.

Finally, the program modifies the data class files, treename file, and sysdata to reflect the new tree structure and exits.

Flow Chart: See following page.


comnod
(# of nodes on next level < 3) or (# of lowest nodes on next level < 2) ?
yes
Print error message
Return
no
Display list of possible nodes to be combined
User input list of nodes to be combined and new nodename
Create new data class file and delete old ones
Update means and covariances
Adjust sysdata
Return

Internal Subroutine Name: conmatsm

Calling Sequence: call conmatsm (ptrs)

Input Parameters:

ptrs (5)ptr ptrs(1) - sysdata
ptrs(2) - scratch
ptrs(3) - display
ptrs(4) - treename
ptrs(5) - mooslogic

Input File Settings: The display file must be set up according to the confusion matrix display file format described previously (section 3.1) In the case of partial pairwise evaluation, some information must also be stored in the scratch file.

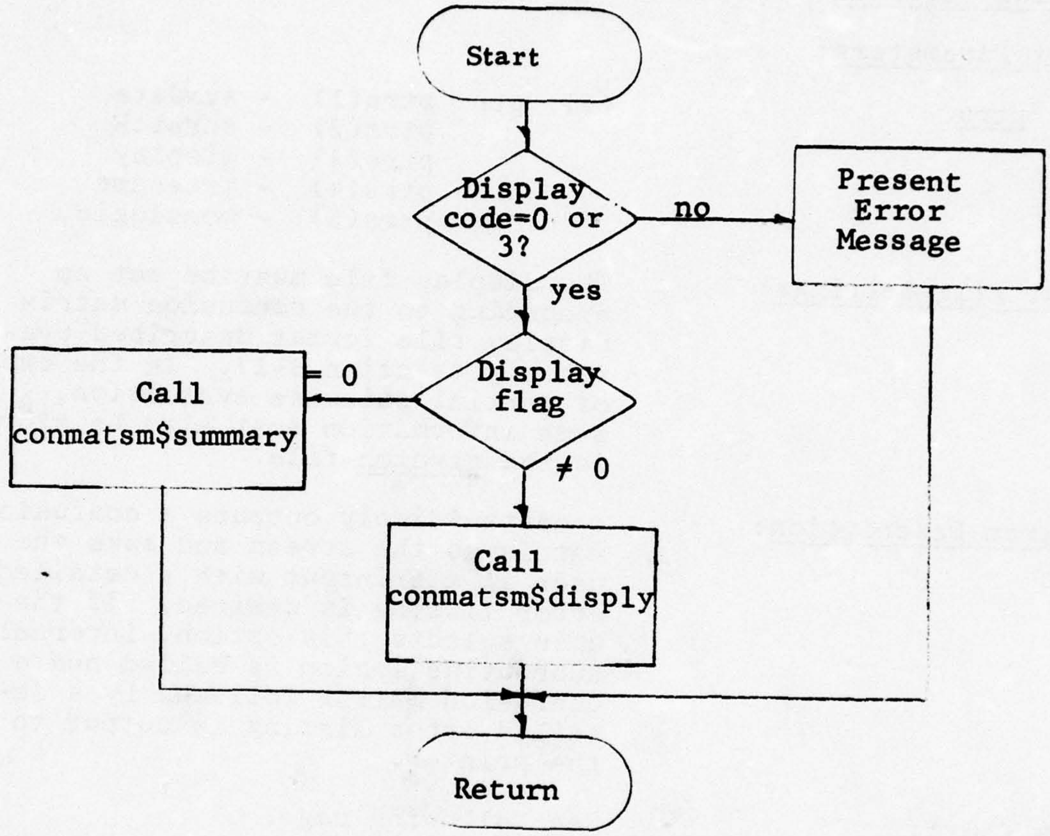
Output File Settings: If conmatsm calls for output to the printer, eval_file is created in the user's login directory and the output is stored there until it can be printed.

The C2, C10, C11, C12, C24, C25, and C26 entries in the display file are set by conmatsm

Program Description: conmatsm checks the display code and calls the appropriate subroutine to output the desired confusion matrix or confusion matrix summary to the correct device.

Flow Chart: See following page

conmatsm



Start
Display code=0 or 3?
no
Present Error Message
yes
Display flag
= 0
Call conmatsm$summary
≠ 0
Call conmatsm$disply
Return

Internal Subroutine Name: conmatsm$disply

Calling Sequence: call conmatsm$disply (ptrs)

Input Parameters:

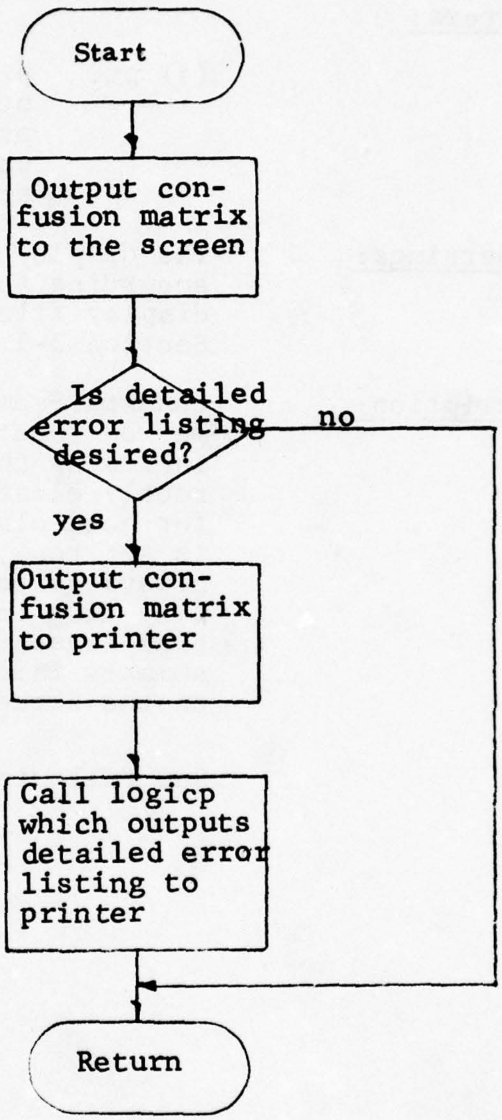
ptrs (5) ptr ptrs(1) - sysdata
ptrs(2) - scratch
ptrs(3) - display
ptrs(4) - treename
ptrs(5) - mooslogic

Input File Settings: The display file must be set up according to the confusion matrix display file format described previously (section 3-1). In the case of partial pairwise evaluation, some information must also be stored in the scratch file.

Program Description: conmatsm$disply outputs a confusion matrix to the screen and asks the user if a printout with a detailed error listing is desired. If the user selects this option, internal subroutine logicp is called and a confusion matrix followed by a detailed error listing is output to the printer.

Flow Chart: See following page.

conmatsm$disply



Internal Subroutine Name: conmatsm$summary

Calling Sequence: call conmatsm$summary (ptrs)

Input Parameters:

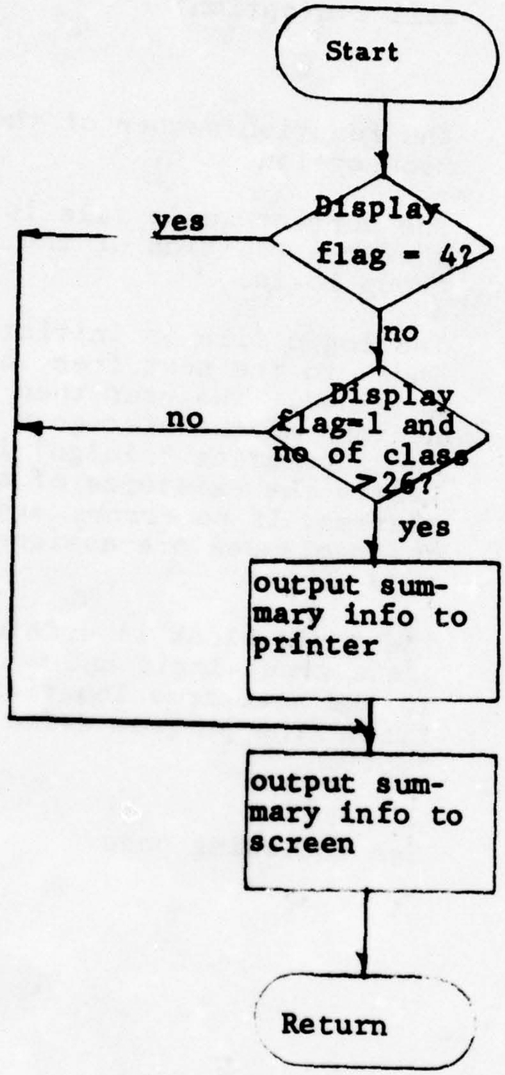
ptrs	(5) ptr	ptrs(1) - sysdata ptrs(2) - scratch ptrs(3) - display ptrs(4) - treename ptrs(5) - mooslogic

Input File Settings: The display file must be set up according to the confusion matrix display file format described in Section 3-1.

Program Description: conmatsm$summary outputs summary information about a confusion matrix, including the overall number of correctly classified vectors and errors for each class. If the display flag is set to 4, this information will always go to the screen. If the display flag is 1, and the number of true classes is greater than 26, the summary information will be displayed on the screen and printed.

Flow Chart: See following page

conmatsm$summary



Internal Subroutine Name: cos

Calling Sequence: call cos(option)

Input Parameters:

option — the function number of the current moos option

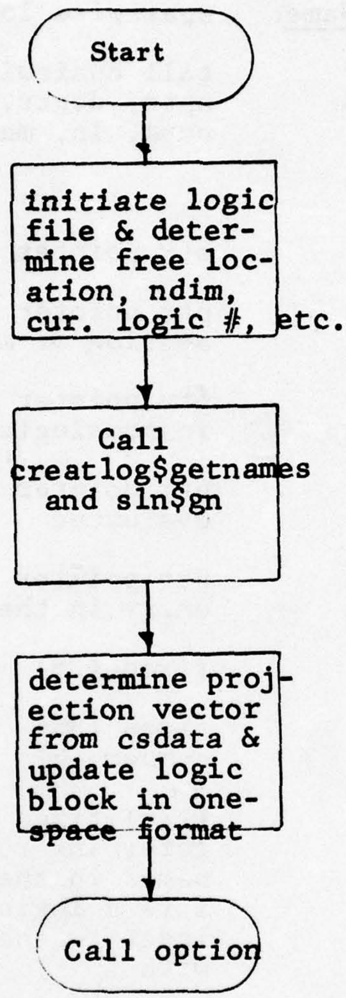
Output File Settings: The current logic file is adjusted to show the addition of the one-space group logic.

Program Description: The logic file is initiated and the index to the next free location is obtained. The user then associates regions of the histogram with classes. The subroutine "sln$gn" is called to verify the existence of all the classes. If no errors are encountered, these classes are assigned a logic node number.

The logic block is updated with one-space group logic and word 5 is set to the next free location in the file. The program exits by calling "option."

Flow Chart: **See following page**

cos


Start
initiate logic file & determine free location, ndim, cur. logic #, etc.
Call creatlog$getnames and sln$gn
determine projection vector from csdata & update logic block in one-space format
Call option

Internal Subroutine Name: cpairwise_logic

Calling Sequence: call cpairwise_logic (lptr, aptr, nptr, dcptr, erptr, ndim, cn, nnum, ln, max, cflag, optr)

Input Parameters:

lptr	ptr pointer to mooslogic file
aptr	ptr pointer to a priori probability section of mooslogic file
nptr	ptr pointer to current logic node in mooslogic file
dcptr	ptr pointer to the vector being evaluated
erptr	ptr pointer to the next available entry in the pair_error_file
ndim	fixed (35) no. of dimensions
cn	fixed (35) current logic node number
nnum	(128) fixed (35) array of indices referring to the table of class names in the mooslogic file. If i is a logic node number, nnum(i) = index to the class name associated with i
ln	(128, 72) fixed (35) If i is a pairwise logic node number, ln(i,1) = the first logic node number beneath i, ln(i, 2) = the next logic node number, etc.
cflag	fixed (35) Usually set to 0. If cflag is set to 1, pairwise_logic outputs vote counts for each classified vector to the user-output stream.

optr — ptr pointer to a file which contains overlap information.

Output Parameters:

max — fixed (35) The vote count at which the vector was classified. This parameter is used by partial pairwise evaluation (pevpw).

cn — fixed (35) The logic node number to which the vector is assigned.

Input File Settings: The word pointed to by "erptr" in the pair_error_file must be set to the logic node number of the true class. The next word in this file must be set to the minimum vote count threshold.

Output File Settings: If a vector is incorrectly classified, tied, or rejected, an entry is created in the pair_error_file for that vector.

Program Description:

cpairwise_logic classifies a given vector using the information stored at a pairwise logic node in a mooslogic file. If the vector is misclassified, tied, or rejected, an entry is made in pair_error_file. cpairwise_logic is used in place of pairwise_logic when partial classification information from a previously performed closed decision boundary logic is available. See write-ups on closed decision boundary logic in Section I.

MOOS Function Name: crdinput

MOOS Function Number: 1

Calling Sequence: Type in "crdinput (treename)"

Input Parameters: A unique 8-character tree name

Output File Settings:

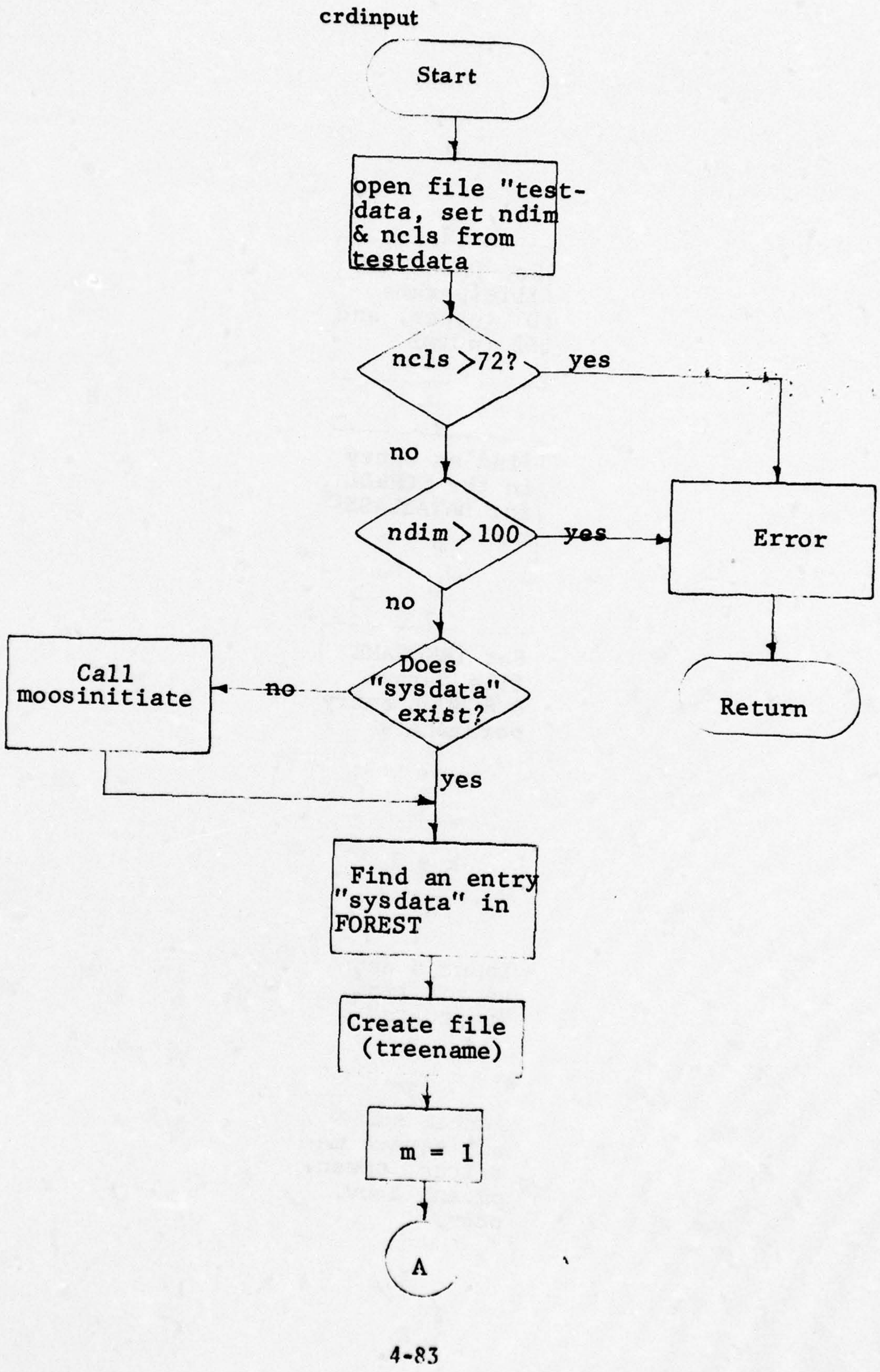
"sysdata" File: Replace a "notatree" entry in the FOREST section with a new tree entry and add an entry to the SCHOOL segment (or replace a "nono" entry) for each apriori node.
Set CSS1 = (treename)
CSS2 = ****
reset CSS4 if appropriate

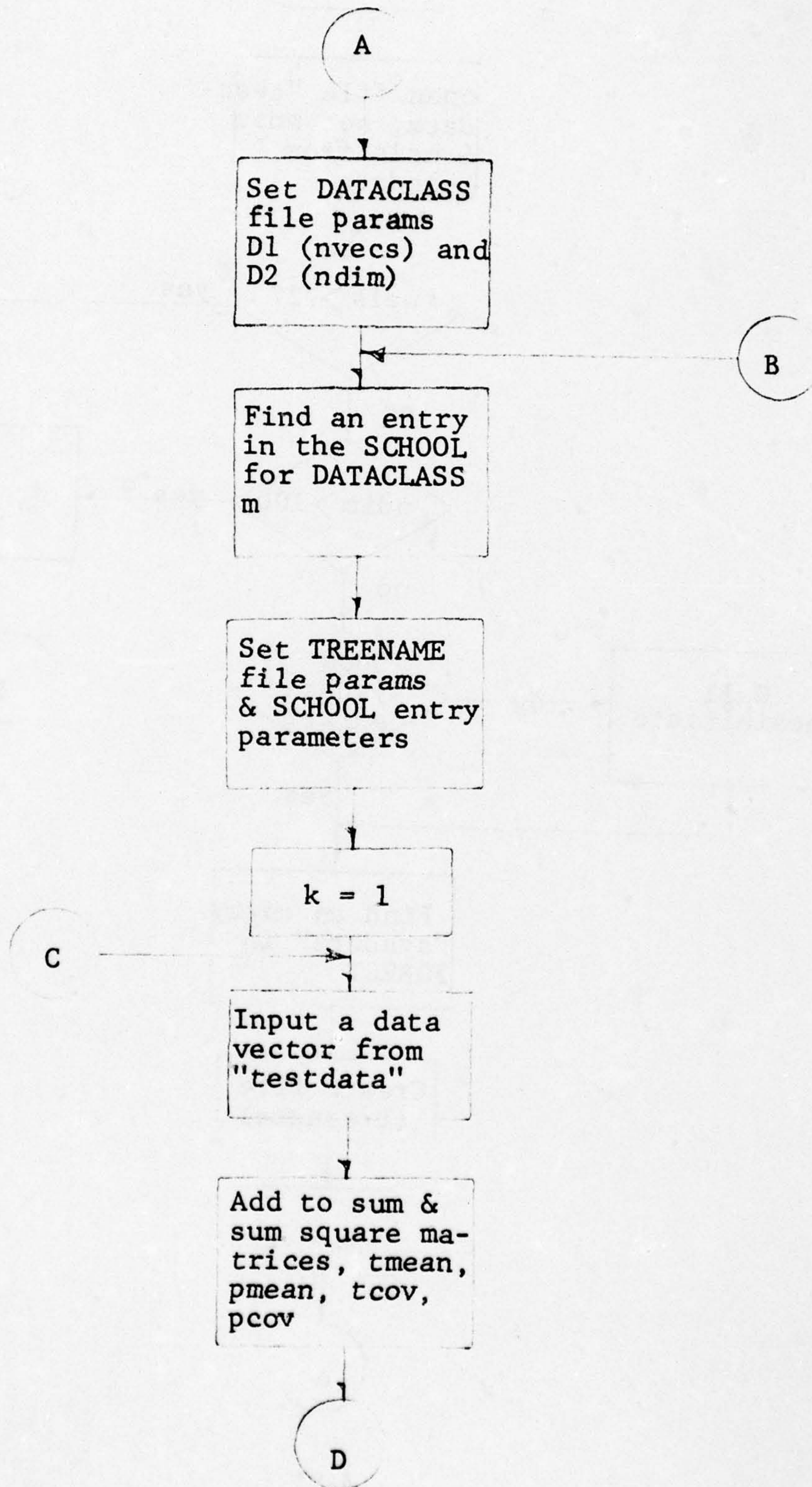
TREENAME File: Create a file under name "treename" and set parameters within the file as appropriate for the input data

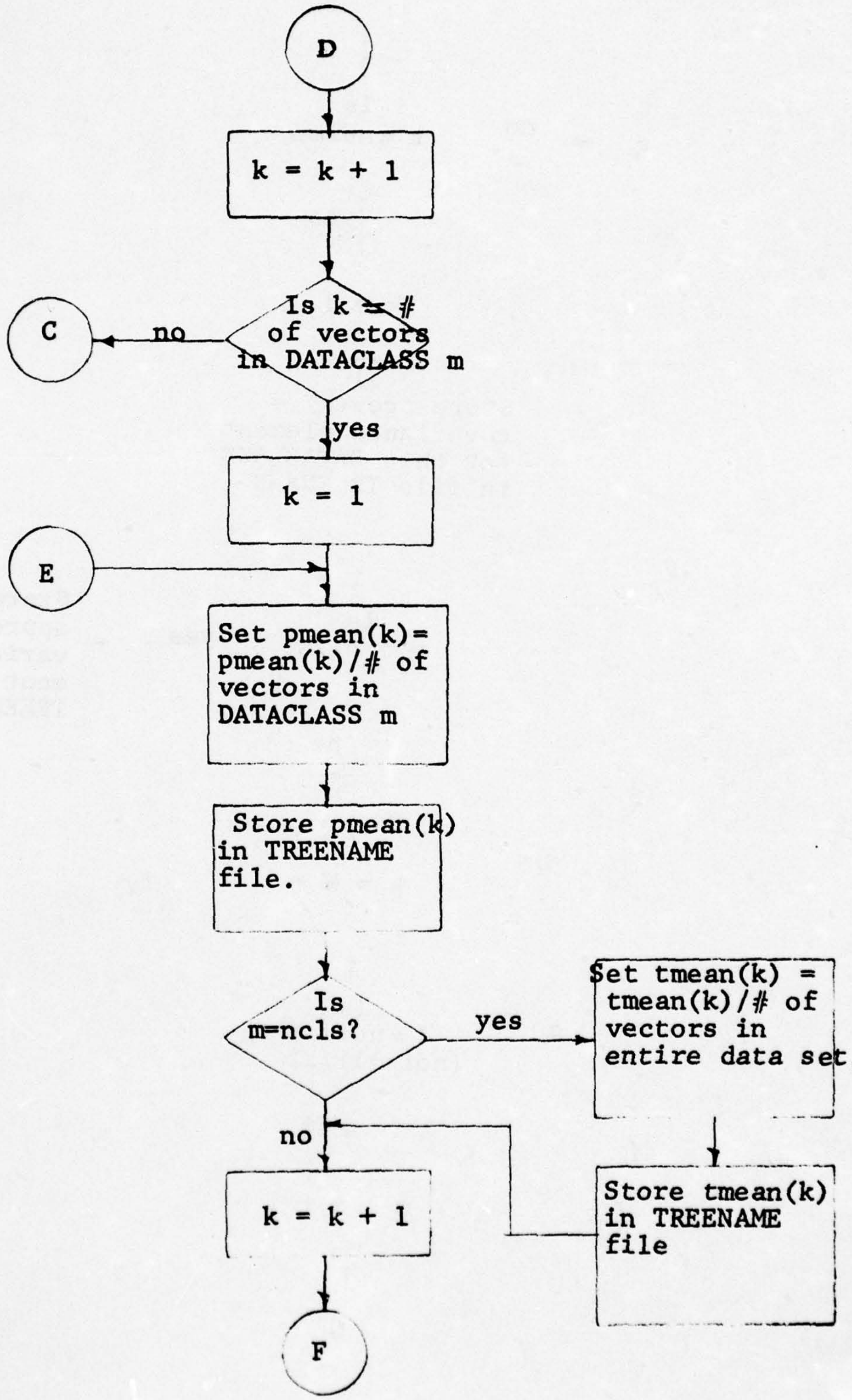
DATACLASS Files: Create a file for each apriori data class and store the appropriate data vectors

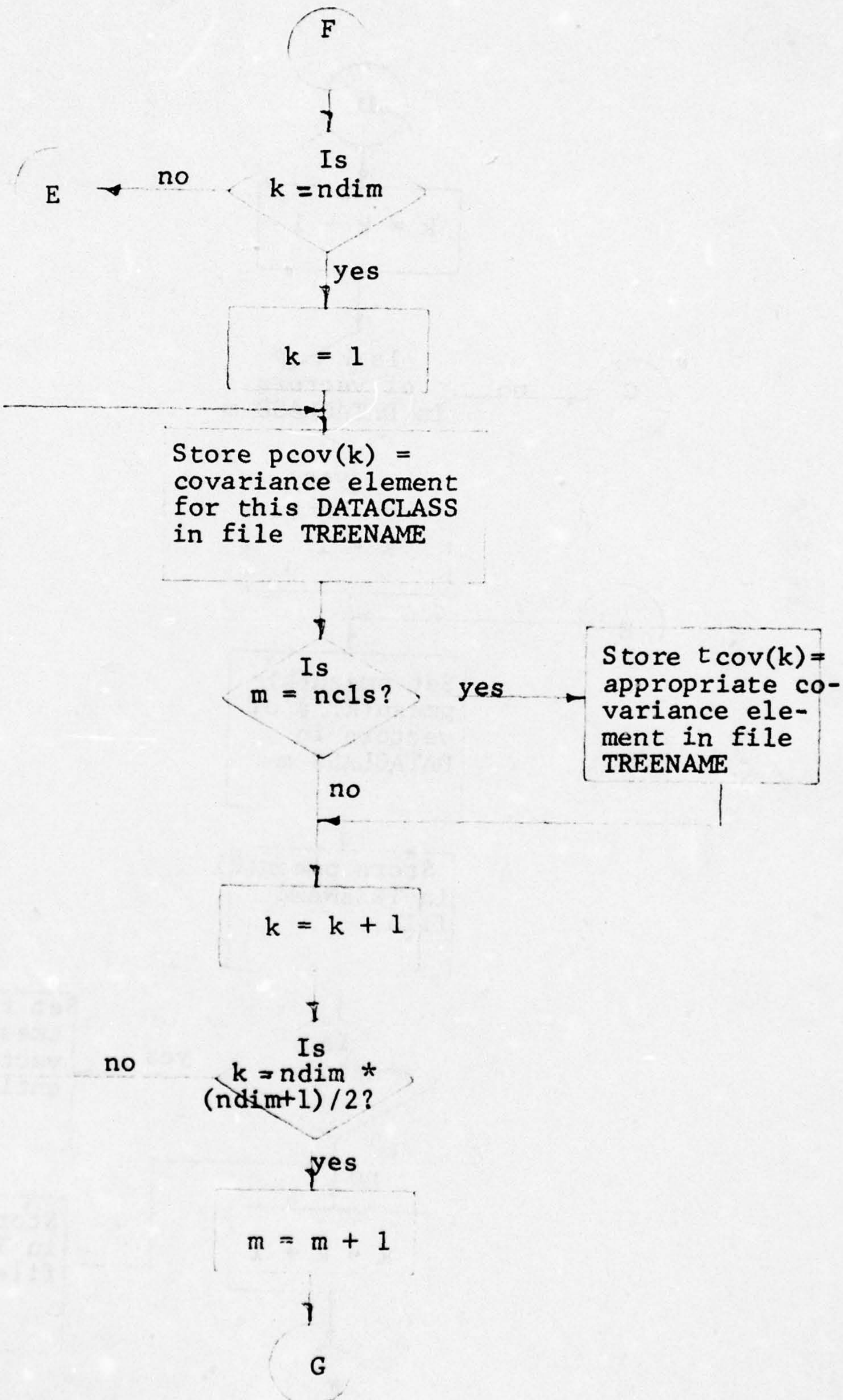
Program Description: "crdinput" transfers data from user's file "testdata" into the user's temporary storage area in standard MOOS data tree format.

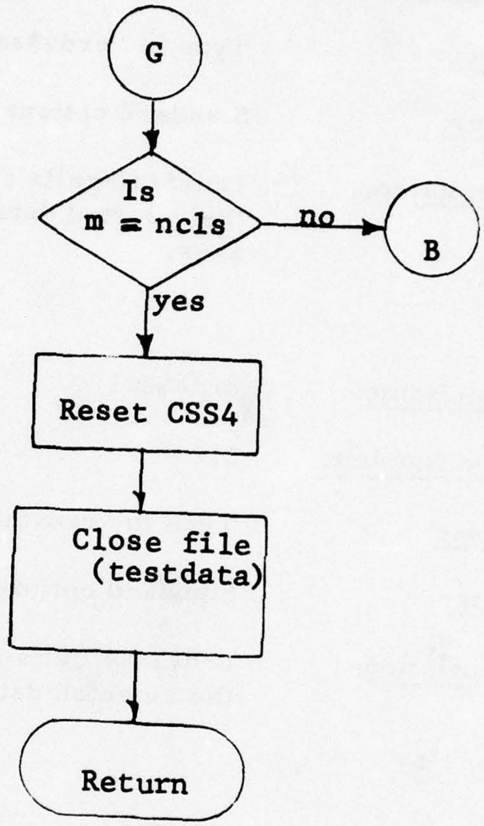
Flow Chart: See following pages


crdinput
Start
open file "test-data, set ndim & ncls from testdata
ncls >72?
yes
no
ndim > 100
yes
Error
Return
no
Does "sysdata" exist?
no
Call moosinitiate
yes
Find an entry "sysdata" in FOREST
Create file (treename)
m = 1
A


A
Set DATACLASS file params D1 (nvecs) and D2 (ndim)
B
Find an entry in the SCHOOL for DATACLASS m
Set TREENAME file params & SCHOOL entry parameters
k = 1
C
Input a data vector from "testdata"
Add to sum & sum square matrices, tmean, pmean, tcov, pcov
D


D
k = k + 1
Is k = # of vectors in DATACLASS m
no
C
yes
k = 1
E
Set pmean(k)= pmean(k)/# of vectors in DATACLASS m
Store pmean(k) in TREENAME file.
Is m=ncls?
yes
Set tmean(k) = tmean(k)/# of vectors in entire data set.
Store tmean(k) in TREENAME file
no
k = k + 1
F


F
Is
k = ndim
no
E
yes
k = 1
Store pcov(k) =
covariance element
for this DATACLASS
in file TREENAME
Is
m = ncls?
yes
Store tcov(k)=
appropriate co-
variance ele-
ment in file
TREENAME
no
k = k + 1
Is
k = ndim *
(ndim+1)/2?
no
yes
m = m + 1
G


G
Is
m = ncls
no
B
yes
Reset CSS4
Close file
(testdata)
Return

MOOS Function Name: crdv$sa2

MOOS Function Number: 197

Calling Sequence: Type in "crdv$sa2 [(treename)] [(nodename)] "

Input Parameter: Standard optional data set selection parameters

Program Description: crdv$sa2 calls crdv$coord which projects the selected data set on any two coordinate axes.

MOOS Function Name: crdv$sa1

MOOS Function Number: 212

Calling Sequence: Type in "crdv$sa1 [(treename)] [(nodename)] "

Input Parameter: Standard optional data set selection parameters

Program Description: crdv$sa1 calls crdv$coord1 which projects the selected data set on any coordinate axis.

MOOS Function Name: crdv$ld2

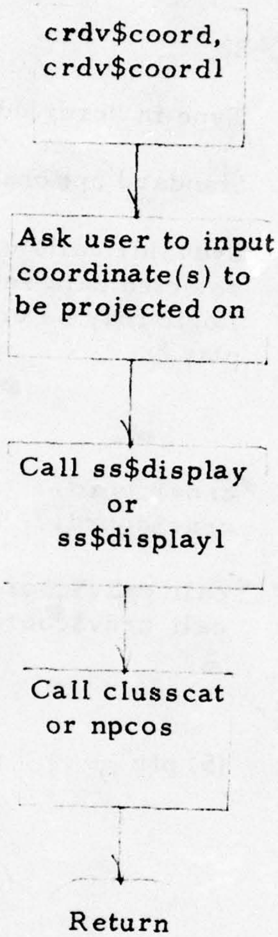
MOOS Function Number: 67

Calling Sequence: Type in "crdv$ld2 [(treename)] [(nodename)] "

Input Parameter: Standard optional data set selection parameters

Program Description: crdv$ld2 calls crdv$coord which projects the selected data set on any two coordinate axes. Logic may be created from the resulting display.

MOOS Function Name:	crdv$ldl
MOOS Function Number:	82
Calling Sequence:	Type in "crdv$ldl [(treename)] (nodename) "
Input Parameters:	Standard optional data set selection parameters
Program Description:	crdv$ldl calls crdv$coordl which projects the selected data set on any coordinate axis. Logic may be created from the resulting display.
Internal Subroutine Name:	crdv$coord crdv$coordl
Calling Sequence:	call crdv$coord (ptrf, x) call crdv$coordl (ptrf, x)
Input Parameters:	
ptrf -	(5) ptr — ptrf(1) - sysdata ptrf(2) - scratch ptrf(3) - display ptrf(4) - treename ptrf(5) - mooslogic
x -	fixed (35) x must be set to 1 for logic design, 0 for structure analysis.
Output File Settings:	Entry coord sets up display for a two-space plot by calling ss$display. Entry coordl sets up csdata for a one-space plot by calling ss$displayl.
Program Description:	coord and coordl allow the user to specify coordinate axes on which to project the selected data set. The display (or csdata) file is then set up and the appropriate display routine called.
Flow Chart:	See following page.


crdv$coord,
crdv$coordl
Ask user to input coordinate(s) to be projected on
Call ss$display or ss$displayl
Call clusscat or npcos
Return

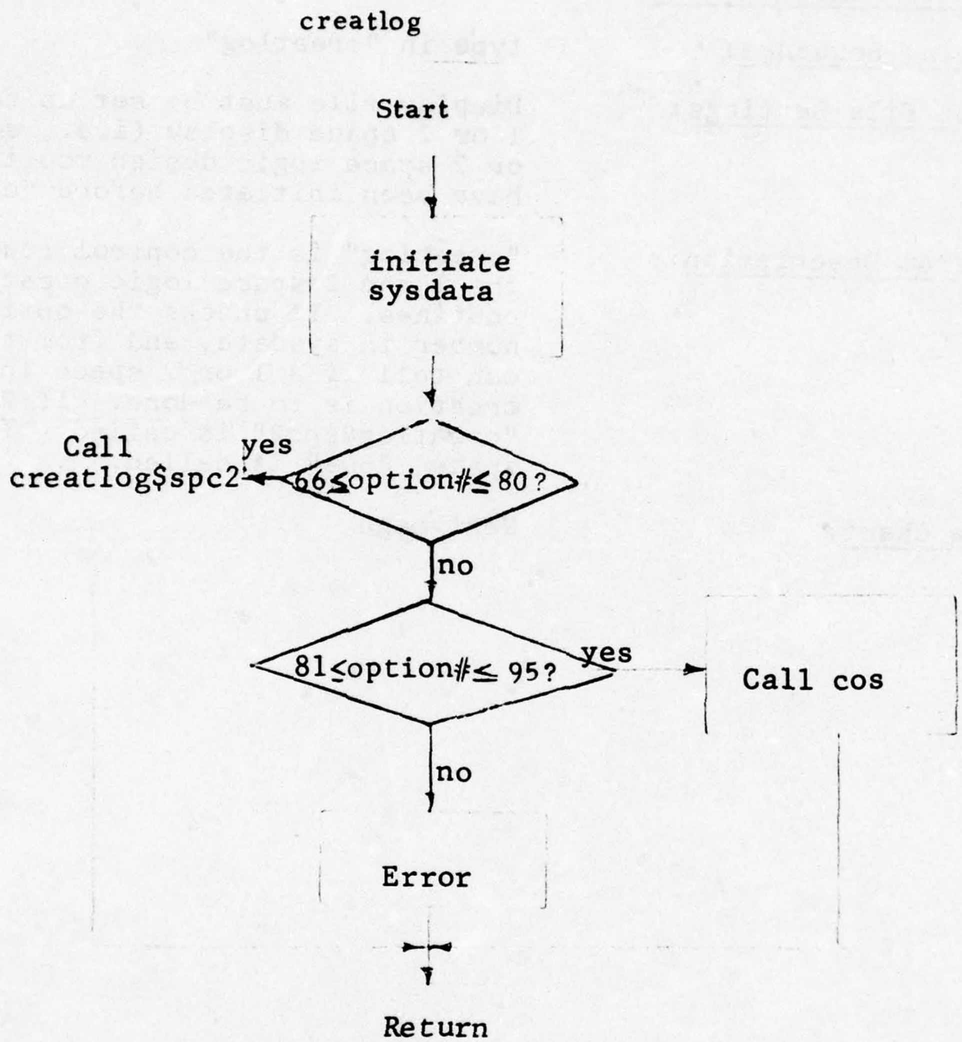
Utility Function Name: creatlog

Calling Sequence: type in "creatlog"

Input File Settings: Display file must be set up for a 1 or 2 space display (i.e., some 1 or 2 space logic design routine must have been initiated before "creatlog")

Program Description: "creatlog" is the control routine for the 1 and 2 space logic creation routines. It checks the option number in sysdata, and from this it can tell if a 1 or 2 space logic creation is to be done. If 2 space, "creatlog$spc2" is called. If 1 space, "cos" is called.

Flow Chart: Next page


creatlog
Start
initiate sysdata
Call creatlog$spc2
yes
66≤option#≤ 80 ?
no
81≤option#≤ 95?
yes
Call cos
no
Error
Return

Internal Subroutine Name: creatlog$discrim

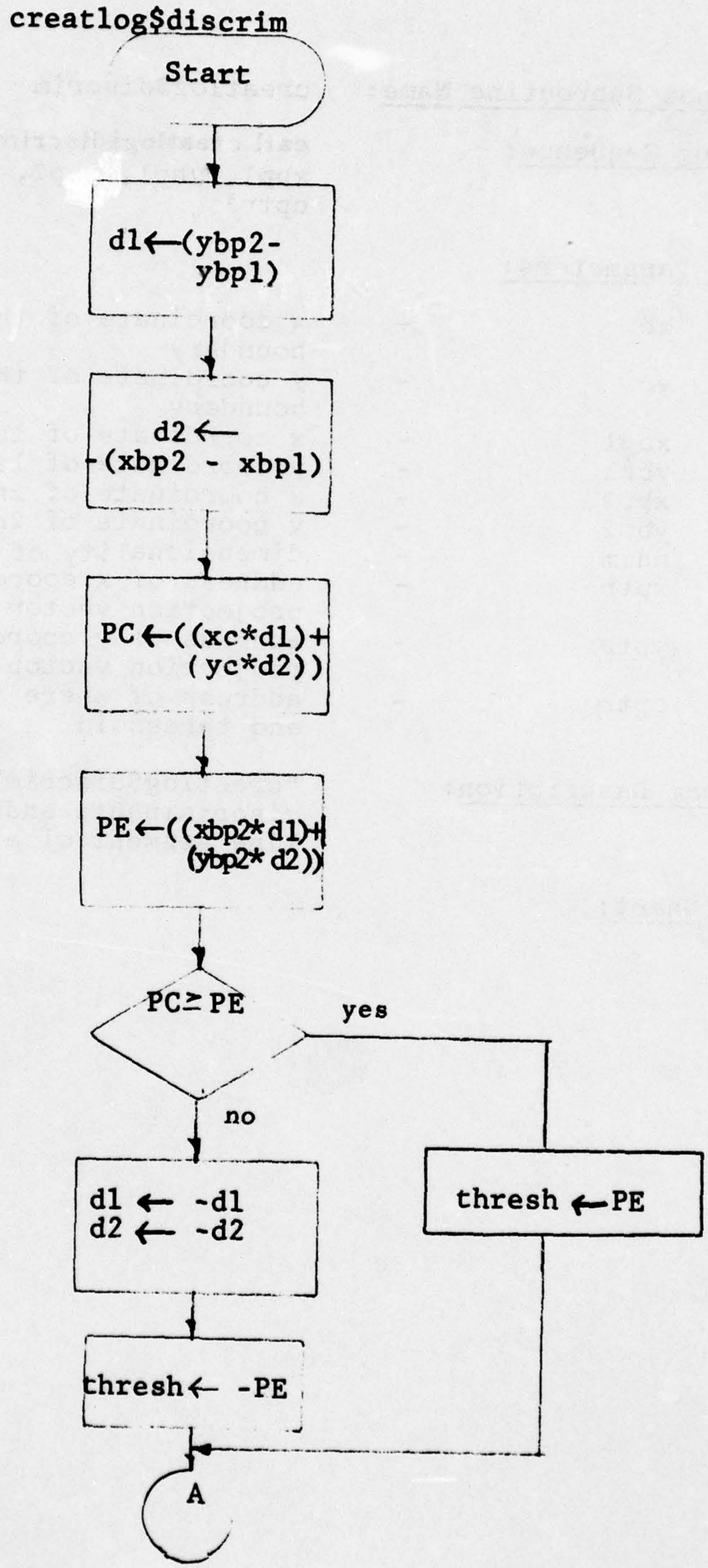
Calling Sequence: **call creatlog$discrim (xc, yc,** xbpl, ybpl, xbp2, ndim, xptr, yptr, cptr)

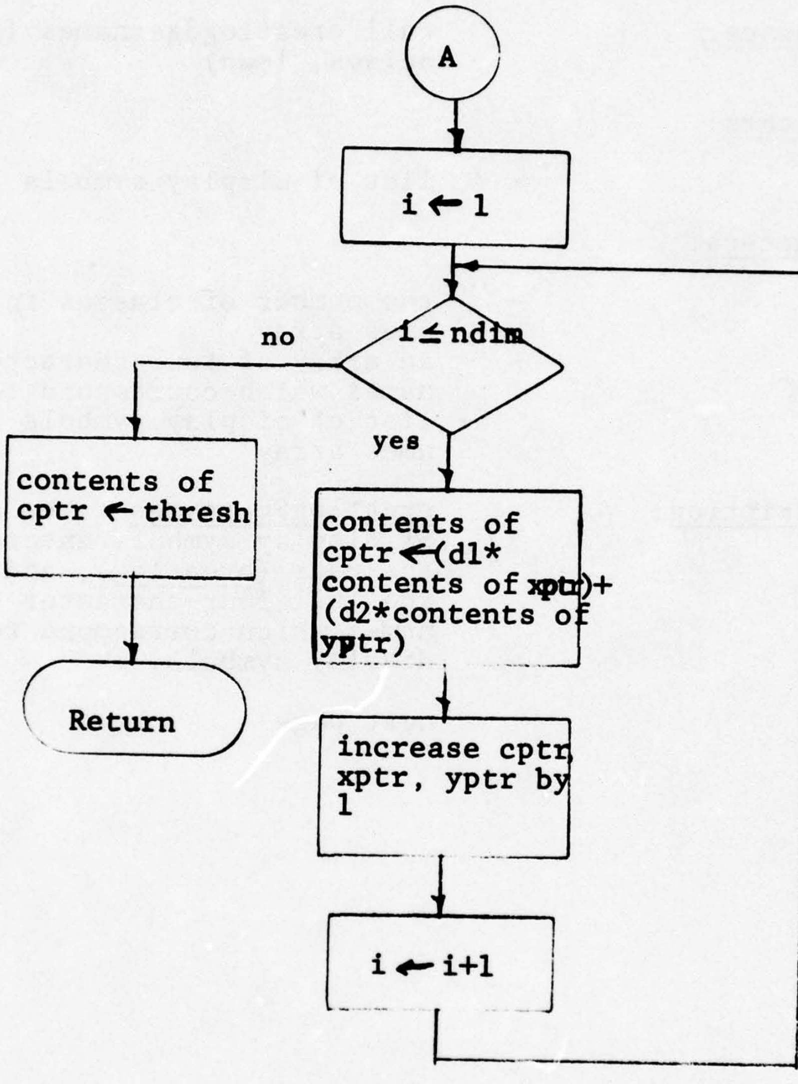
Input Parameters:

xc	-	x coordinate of the convex side of the boundary
yc	-	y coordinate of the convex side of boundary
xbpl	-	x coordinate of 1st point of boundary
ybpl	-	y coordinate of 1st point of boundary
xbp2	-	x coordinate of 2nd point of boundary
ybp2	-	y coordinate of 2nd point of boundary
ndim	-	dimensionality of data
xptr	-	address of x coordinate of the projection vector
yptr	-	address of y coordinate of the projection vector
cptr	-	address of where to place discriminant and threshold

Program Description: "creatlog$discrim" calculates the discriminants and threshold for each line segment of a boundary.

Flow Chart:


creatlog$discrim
Start
d1←(ybp2-ybp1)
d2←-(xbp2 - xbp1)
PC←((xc*d1)+(yc*d2))
PE←((xbp2*d1)+(ybp2*d2))
PC≥PE
yes
no
d1 ← -d1
d2 ← -d2
thresh ←PE
thresh← -PE
A



A
i ← 1
i ≤ ndim
no
yes
contents of cptr ← thresh
Return
contents of cptr ←(d1* contents of xptr)+ (d2*contents of yptr)
increase cptr, xptr, yptr by 1
i ← i+1

Internal Subroutine Name: creatlog$getnames

Calling Sequence: call creatlog$getnames (nmes, nclses, lows)

Input Parameters:

nmes - list of display symbols

Output Parameters:

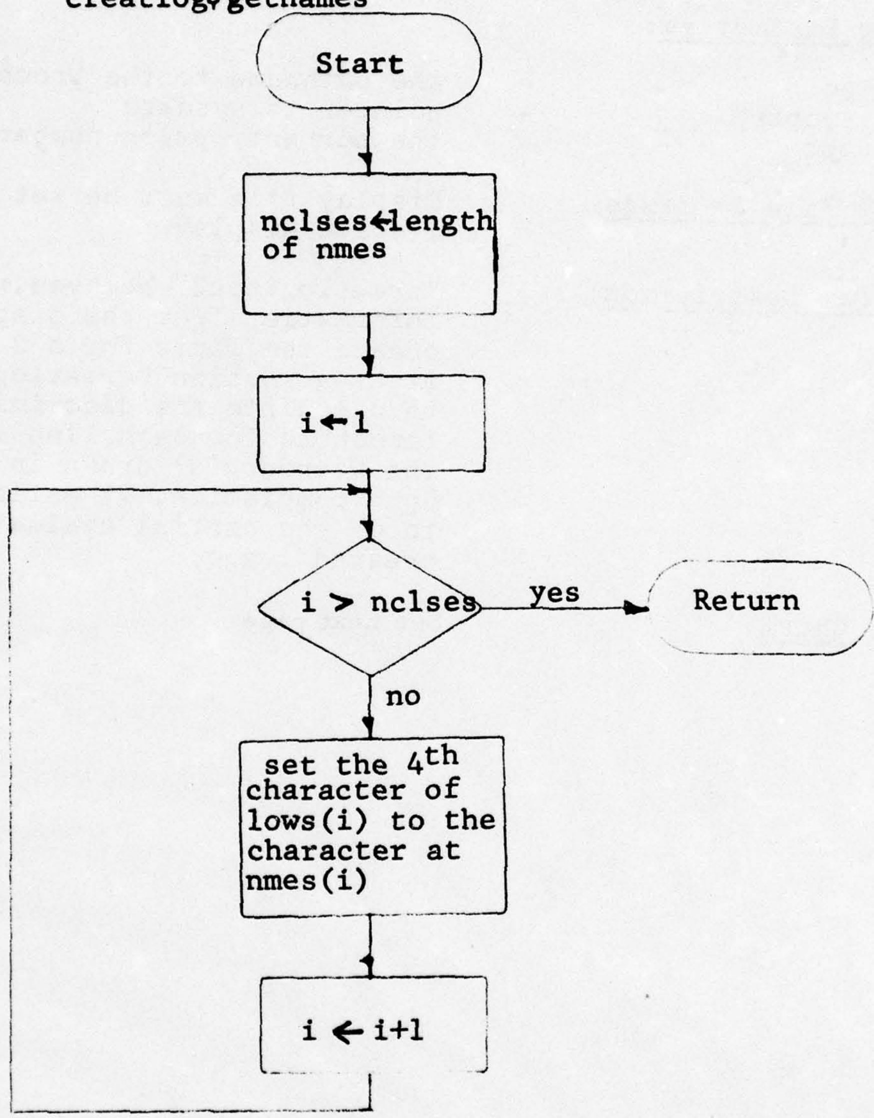
nclses - the number of classes in the nmes array

lows - an array of four-character node names which correspond to the list of display symbols in the nmes array

Program Description: creatlog$getnames takes a list of display symbols entered by the user (creatlog), and returns the full four-character class names which correspond to these display symbols.

Flow Chart: next page

creatlog$getnames



Internal Subroutine Name: creatlog$spc2

Calling Sequence: call creatlog$spc2 (pd, sysptr, opt)

Input Parameters:

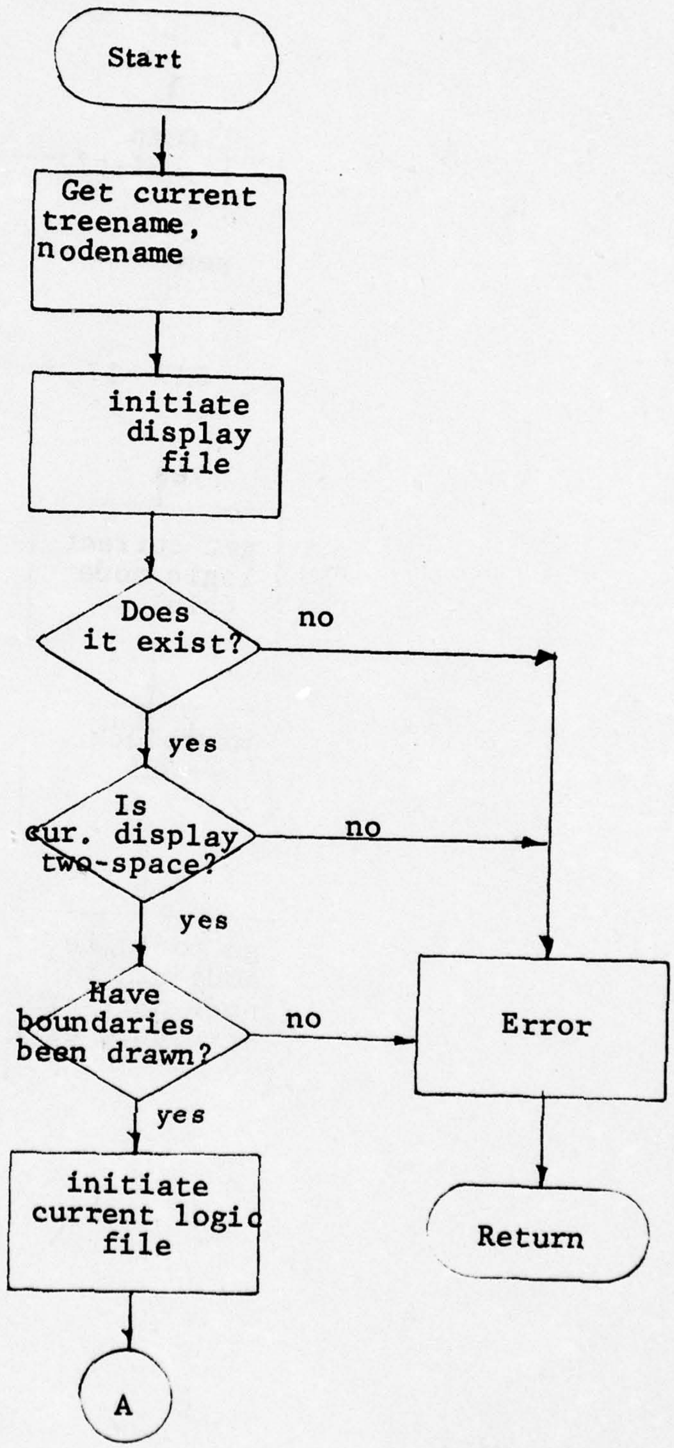
pd	-	the pathname to the process directory
sysptr	-	pointer to sysdata
opt	-	the current option number

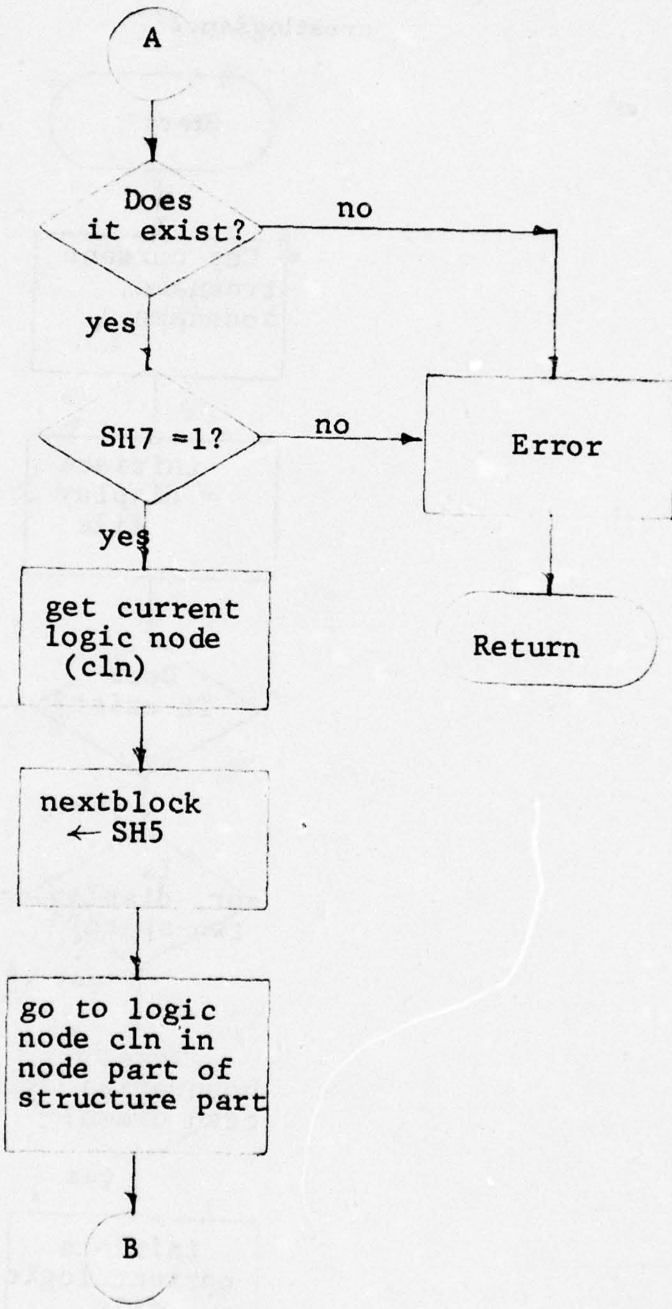
Input File Settings: Display file must be set up for a 2 space display

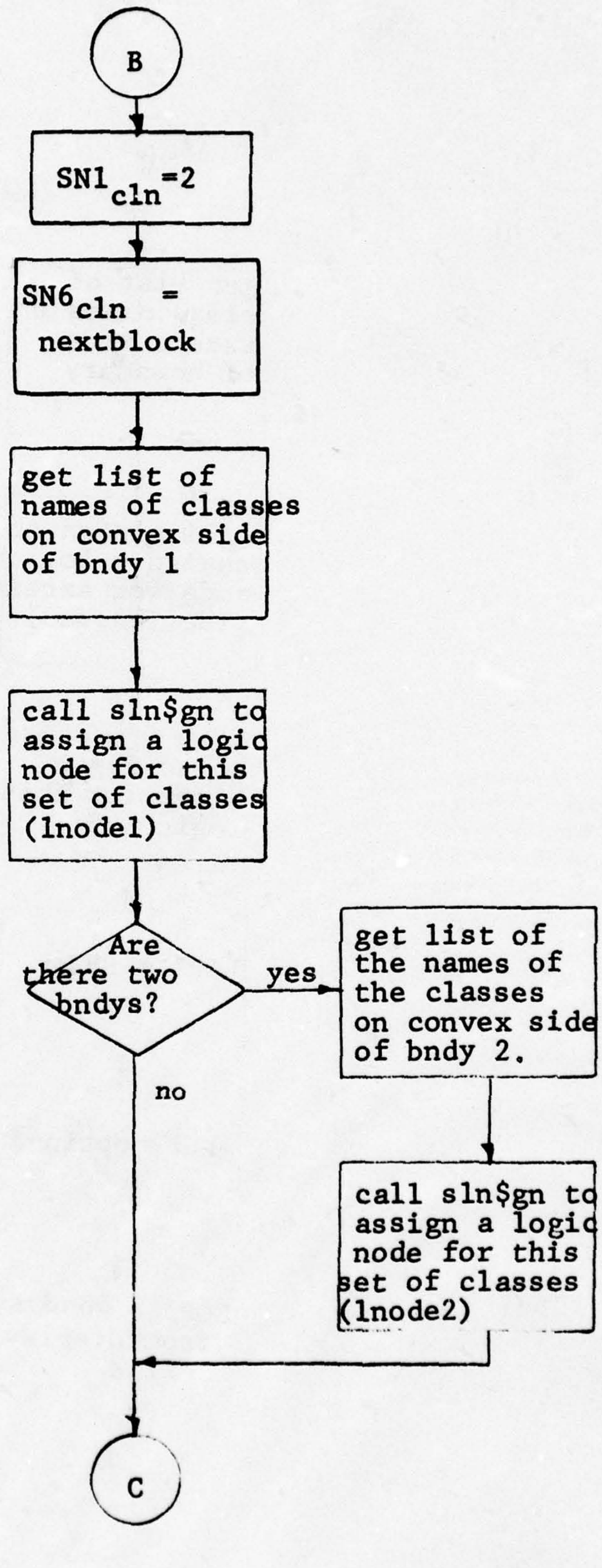
Program Description: "creatlog$spc2" removes the necessary information from the display file to create the logic for a 2 space display. It uses routine "creatlog$discrim" to calculate the discriminants and threshold for each line segment of the boundary(s) drawn in 2 space. Upon completion, it calls "pevgl" to do the partial evaluation of the created logic.

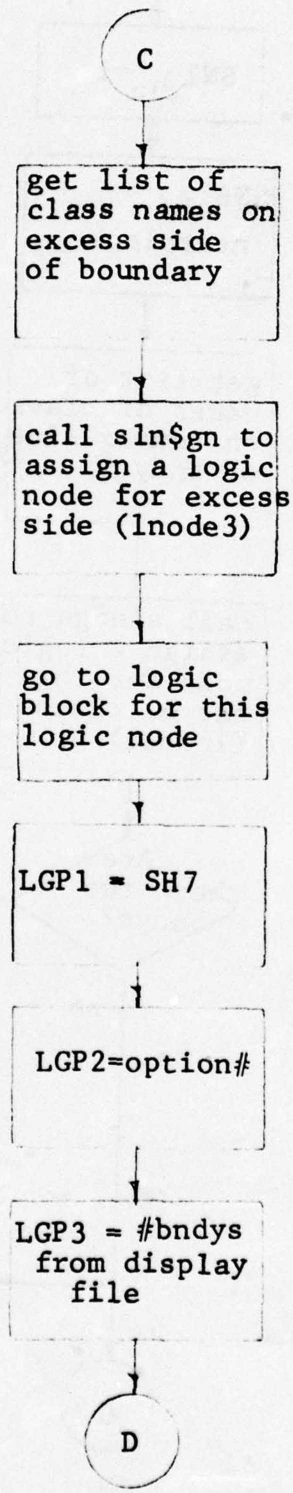
Flow Chart: See next page

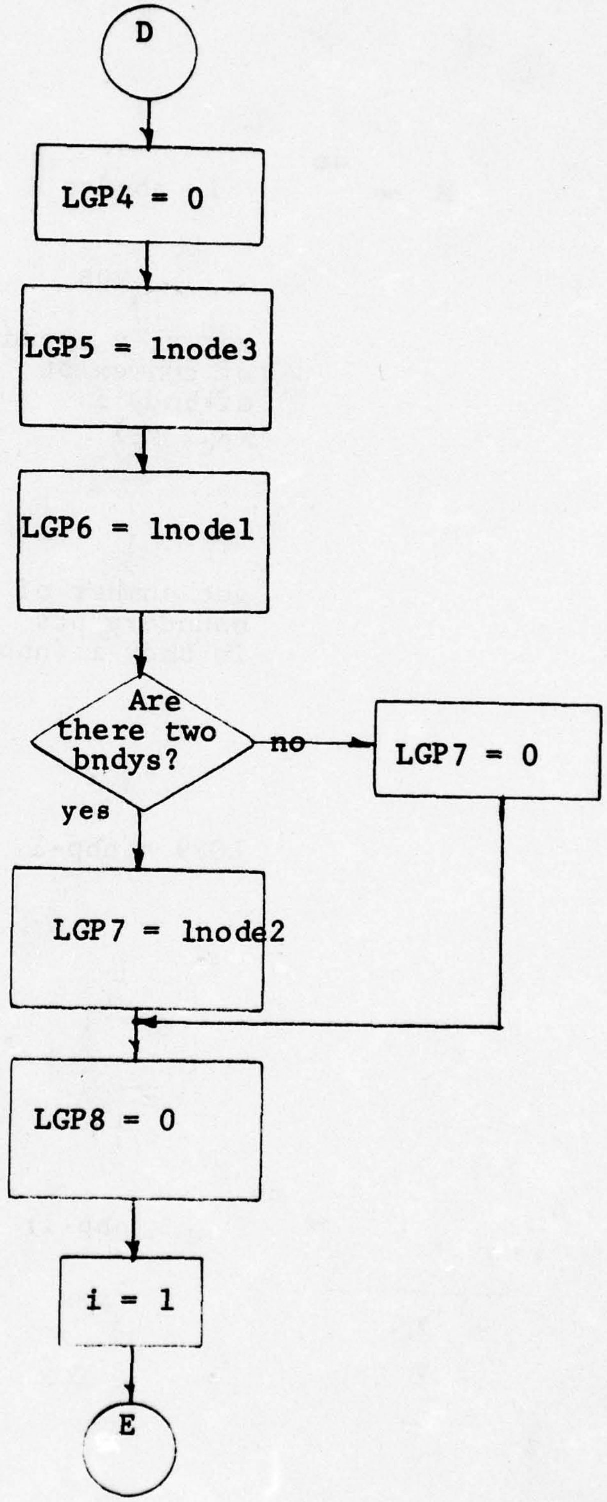
creatlog$spc2

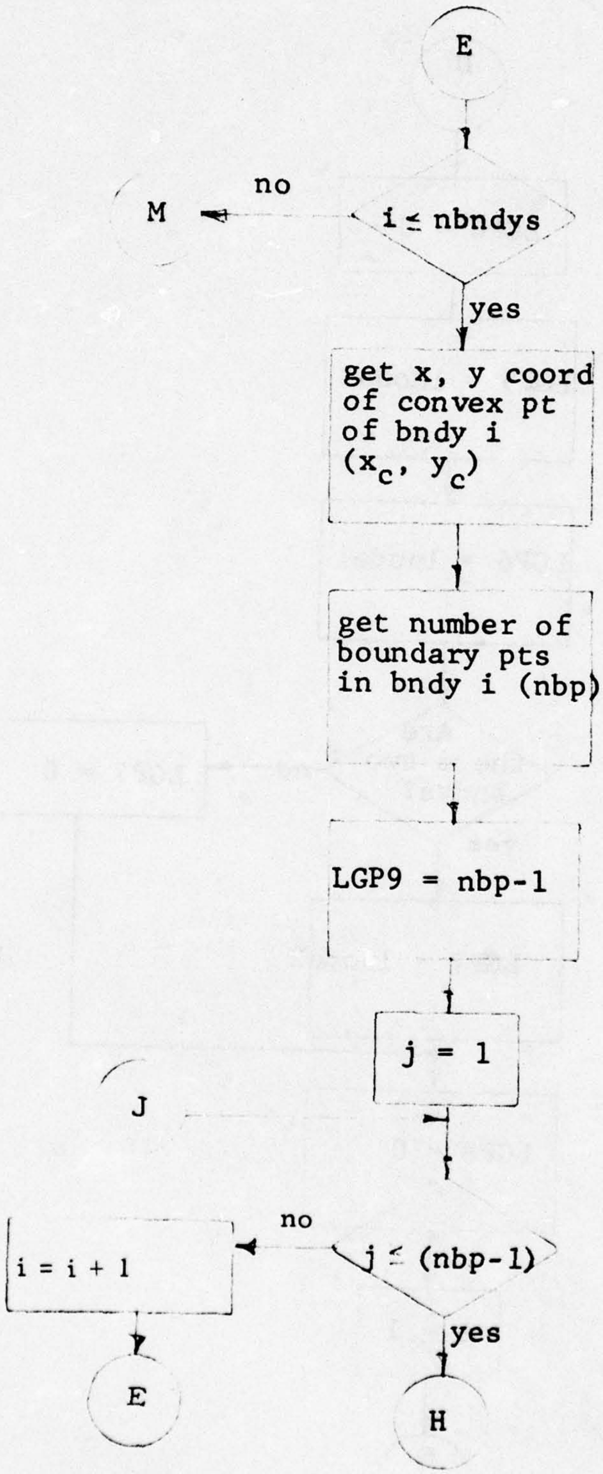


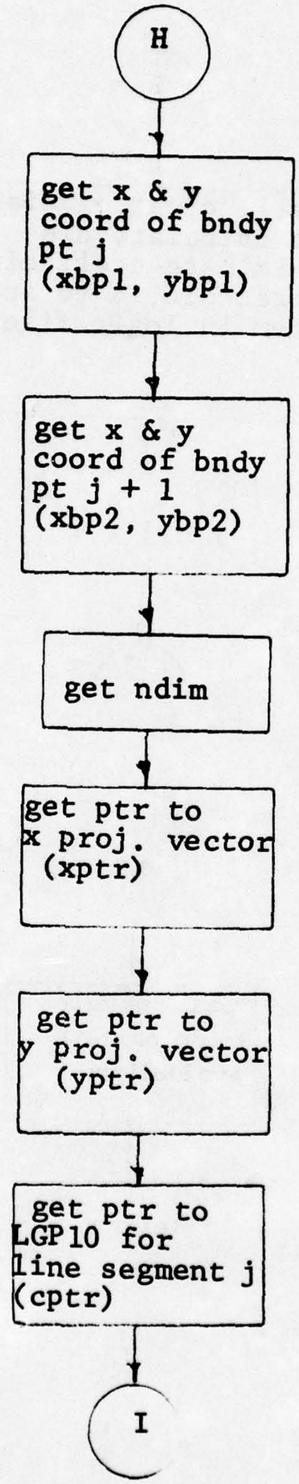

A
Does
it exist?
no
yes
SH7 =1?
no
Error
yes
get current
logic node
(cln)
Return
nextblock
← SH5
go to logic
node cln in
node part of
structure part
B

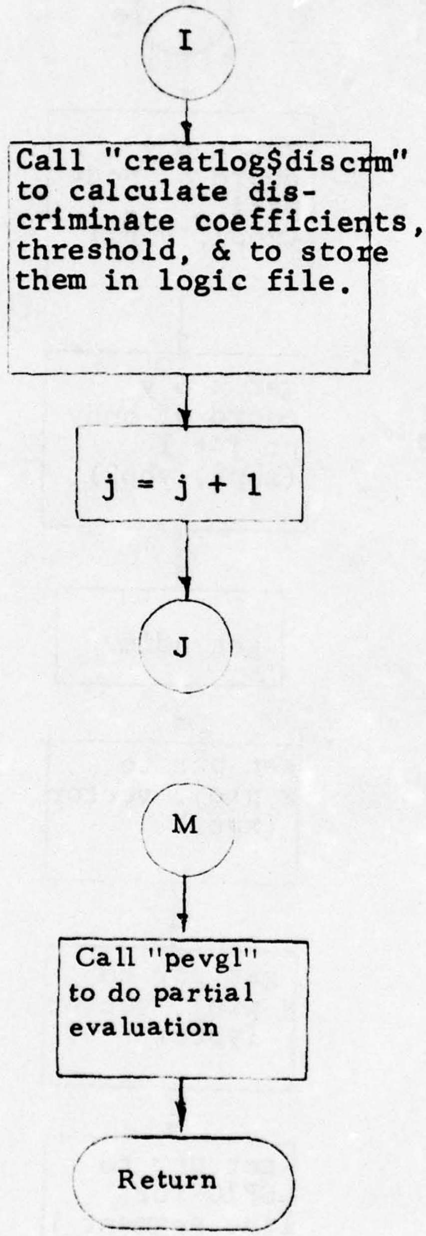

B
SN1cln=2
SN6cln =
nextblock
get list of
names of classes
on convex side
of bndy 1
call sln$gn to
assign a logic
node for this
set of classes
(lnode1)
Are
there two
bndys?
yes
no
get list of
the names of
the classes
on convex side
of bndy 2.
call sln$gn to
assign a logic
node for this
set of classes
(lnode2)
C


C
get list of class names on excess side of boundary
call sln$gn to assign a logic node for excess side (lnode3)
go to logic block for this logic node
LGP1 = SH7
LGP2=option#
LGP3 = #bndys from display file
D


D
LGP4 = 0
LGP5 = lnode3
LGP6 = lnode1
Are there two bndys?
no
LGP7 = 0
yes
LGP7 = lnode2
LGP8 = 0
i = 1
E


E
i ≤ nbndys
no
M
yes
get x, y coord of convex pt of bndy i (x_c, y_c)
get number of boundary pts in bndy i (nbp)
LGP9 = nbp-1
j = 1
J
j ≤ (nbp-1)
no
i = i + 1
E
yes
H


H
get x & y coord of bndy pt j (xbp1, ybp1)
get x & y coord of bndy pt j + 1 (xbp2, ybp2)
get ndim
get ptr to x proj. vector (xptr)
get ptr to y proj. vector (yptr)
get ptr to LGP10 for line segment j (cptr)
I


I
Call "creatlog$discrm" to calculate discriminate coefficients, threshold, & to store them in logic file.
j = j + 1
J
M
Call "pevgl" to do partial evaluation
Return

Moos Function Name: creatree

Moos Function Number: 3

Calling Sequence: type in "creatree (newtree)"

Input Parameters: the standard optional data set selection parameter

Output File Settings: "sysdata" reflects the addition of a new tree in the system. newtree file is created, and appropriate values are inserted.

Data class files are created for each node, and the appropriate vectors are stored in each.

Program Description: creatree creates (newtree) according to user input options. The user indicates the number of nodes in (newtree): 0 indicates merge, < 0 indicates combine, and > 0 indicates a specification. The user next states the number and names of the trees from which newtree is to be created. The user is then asked if vector id's should be sequenced, and if he wishes a listing of these changes. If a listing is desired or the combine option is invoked, an output file create_file is created in his working directory. Lnodes is then called for each tree specified to be used in creating treename.

Next, the scratch file is set up to indicate how (newtree) is to be formed. If the option is combine, the display characters are checked for uniqueness and changed if required. If the option is specify, user interaction specifies how the tree is to be constructed. If the option is merge, the trees are checked for similarity of names.

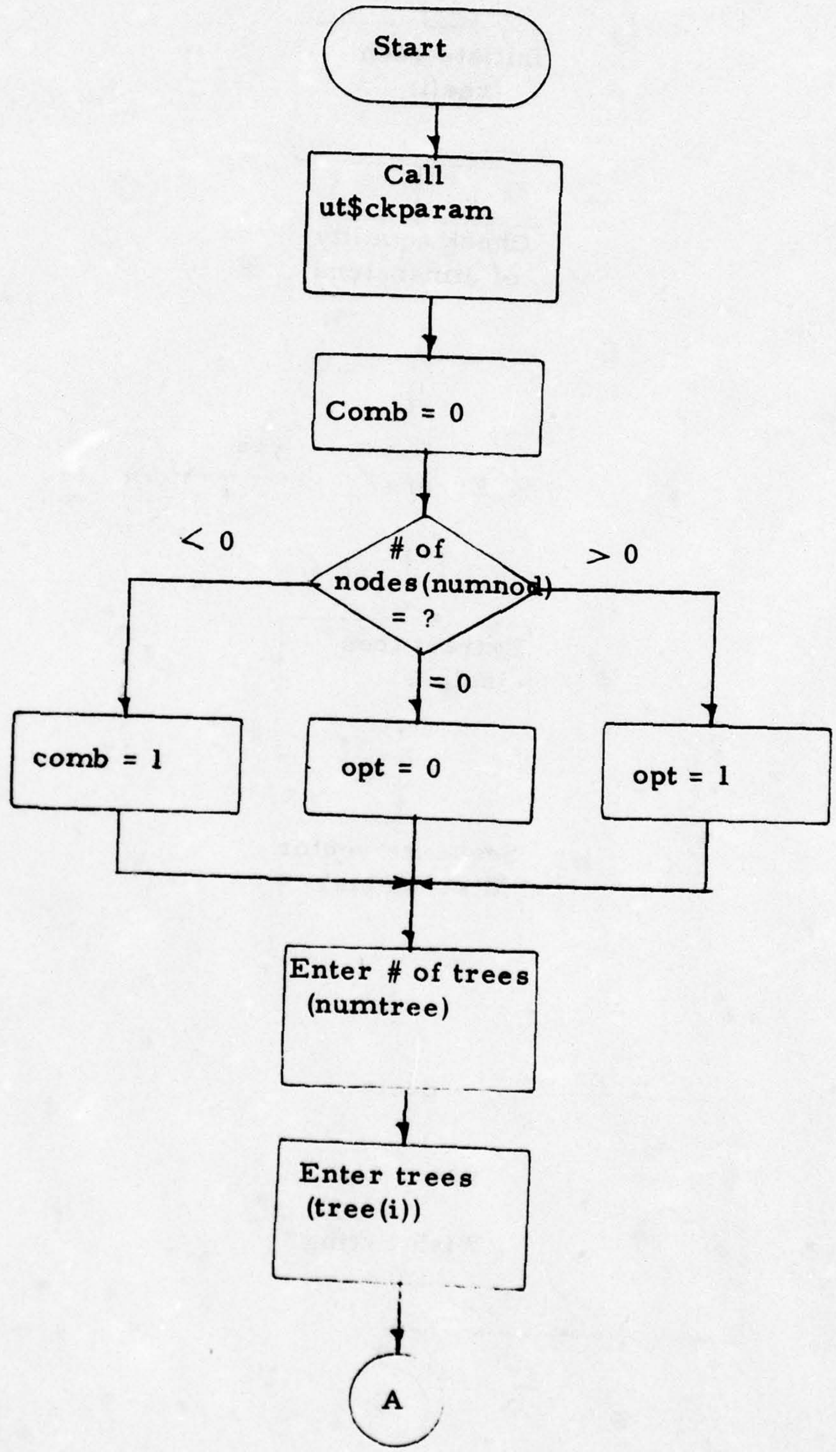
According to the mapping in scratch, the vectors in the appropriate nodes are copied into the correct dataclass files, the treename file is set up by calls to mmeanacv for the senior node and the node being processed, and sysdata is adjusted.

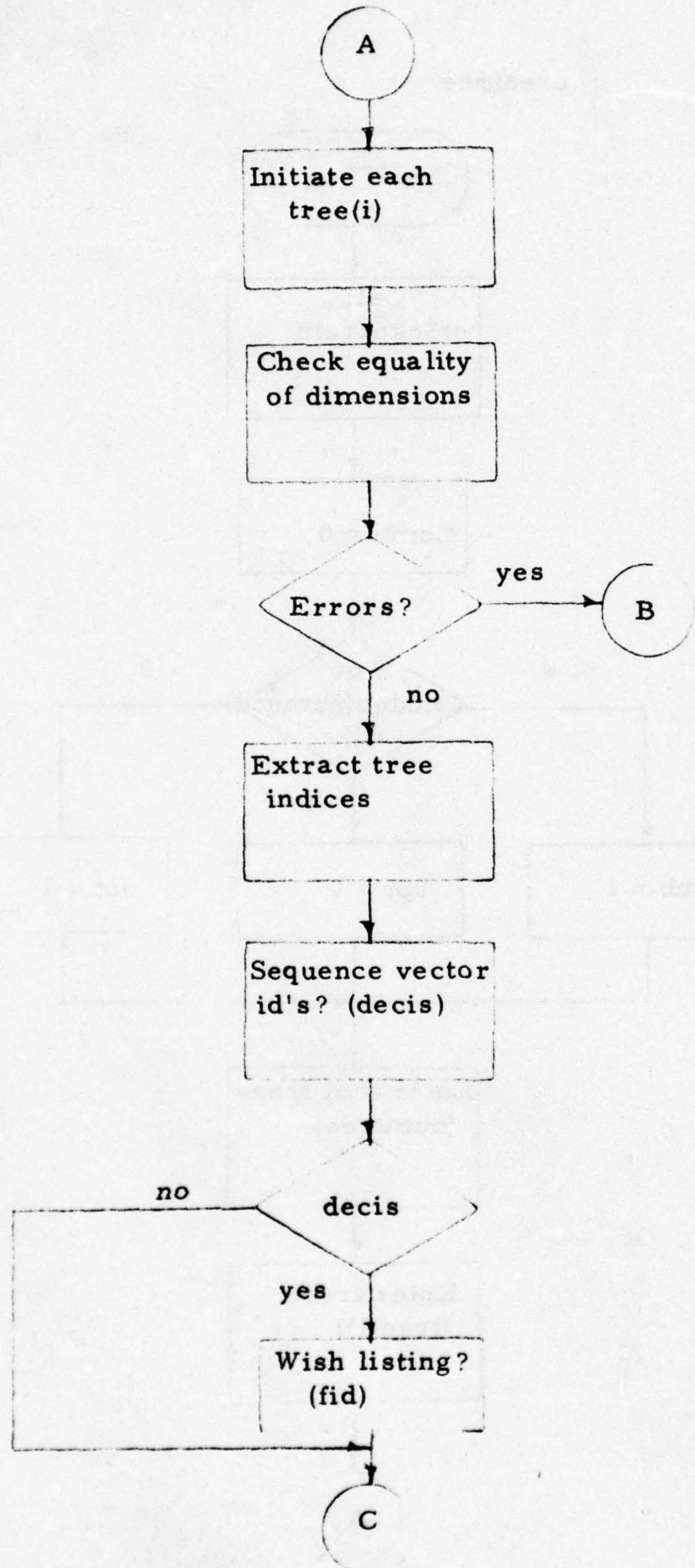
If any errors occur, (newtree) is deleted. If no information is put into create_file, it is deleted. The trees used in forming (newtree) are not affected by this routine.

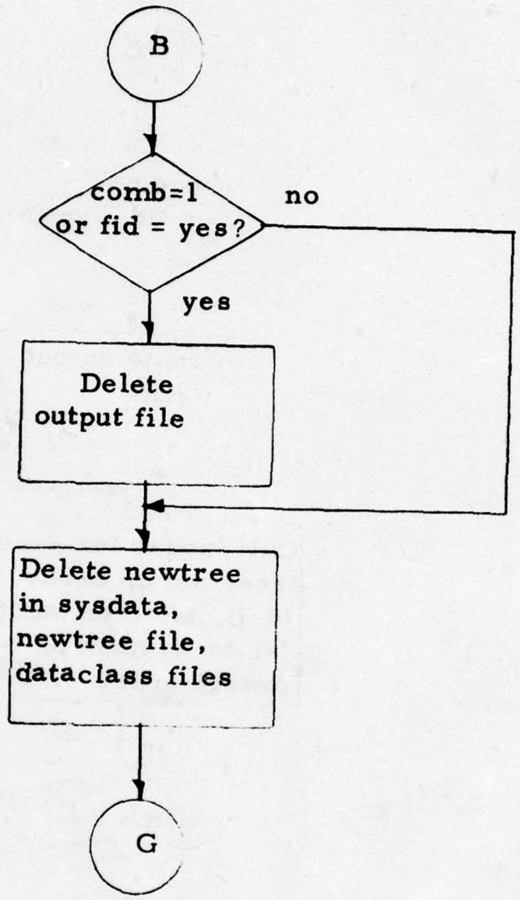
creatree scratch file:	1	C1	C1 number of nodes
		C2(1)	C2(i) number of nodes
		C3(1)	from tree(i) followed
		C3(2)	by node names C3(j)
		⋮	
		C3(C2(1))	
		C2(2)	
		C3(1)	
		C3(2)	
		⋮	
		C3(C2(2))	
		⋮	

Flow Chart: See following page

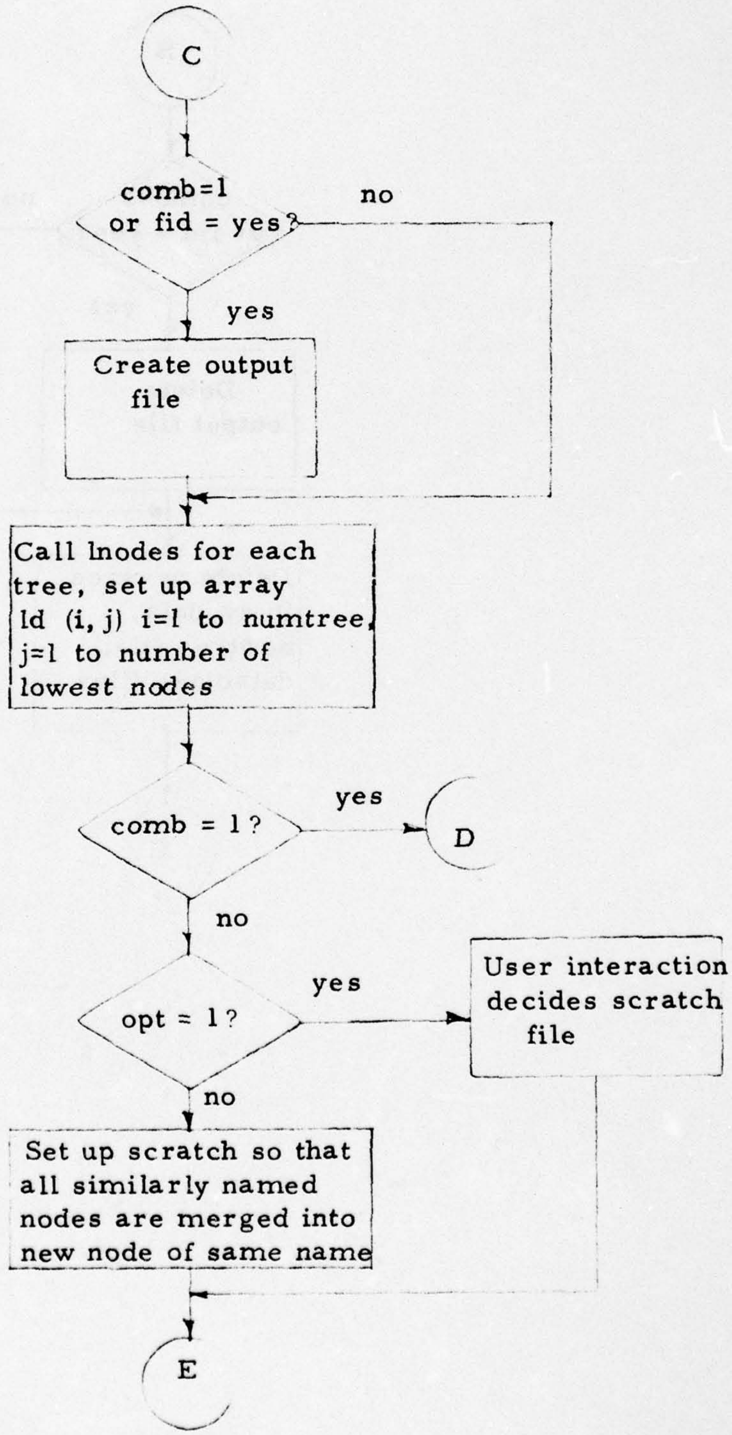
createe

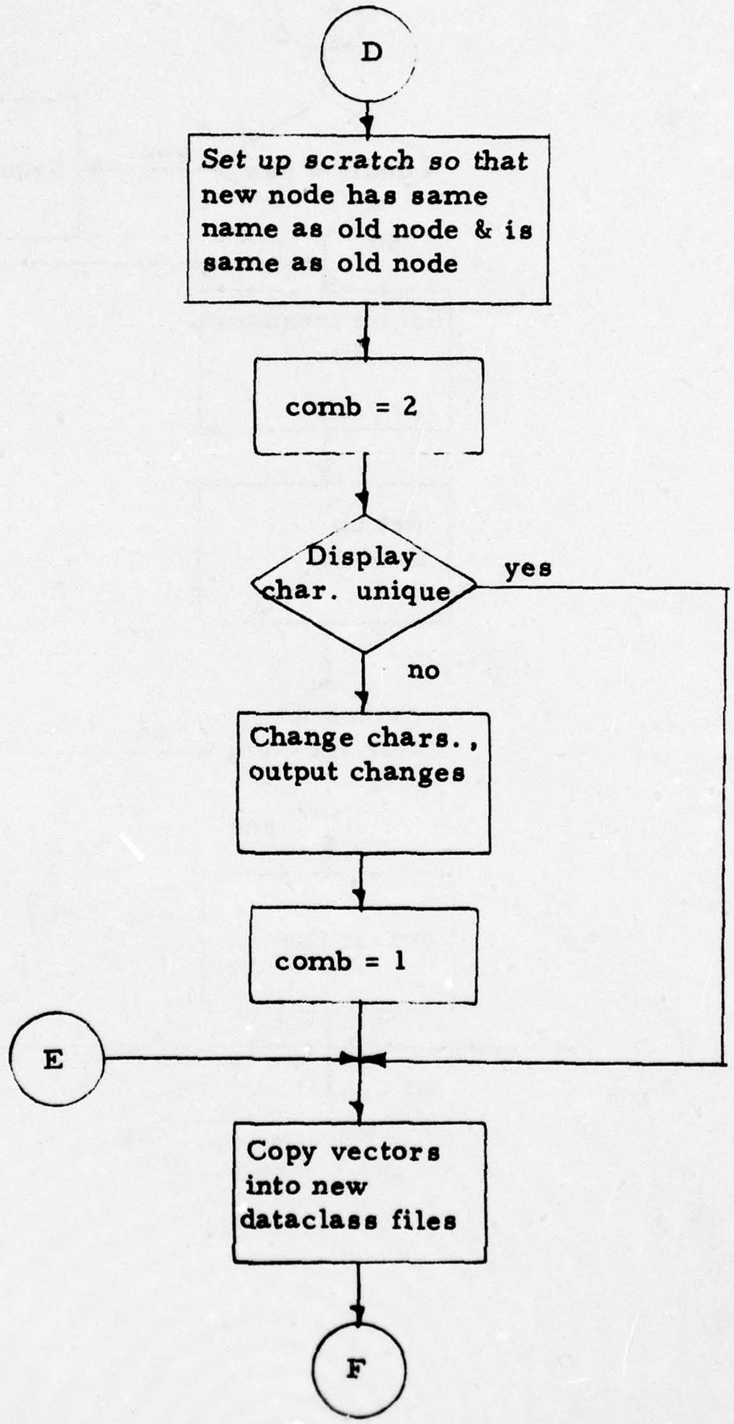

Start
Call
ut$ckparam
Comb = 0
< 0
# of
nodes(numnod)
= ?
> 0
= 0
comb = 1
opt = 0
opt = 1
Enter # of trees
(numtree)
Enter trees
(tree(i))
A

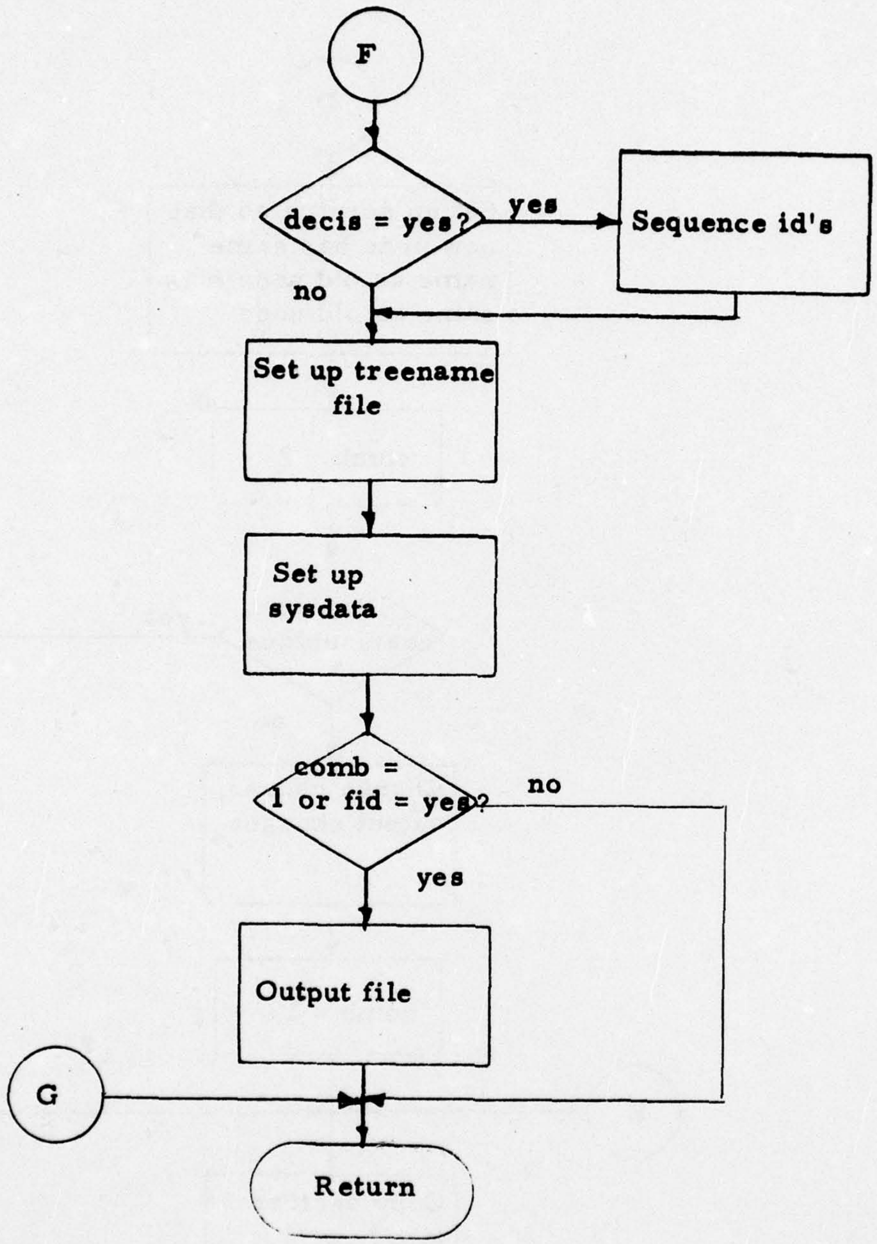

A
Initiate each tree(i)
Check equality of dimensions
Errors?
yes
B
no
Extract tree indices
Sequence vector id's? (decis)
decis
no
yes
Wish listing? (fid)
C



B
comb=1
or fid = yes?
no
yes
Delete
output file
Delete newtree
in sysdata,
newtree file,
dataclass files
G

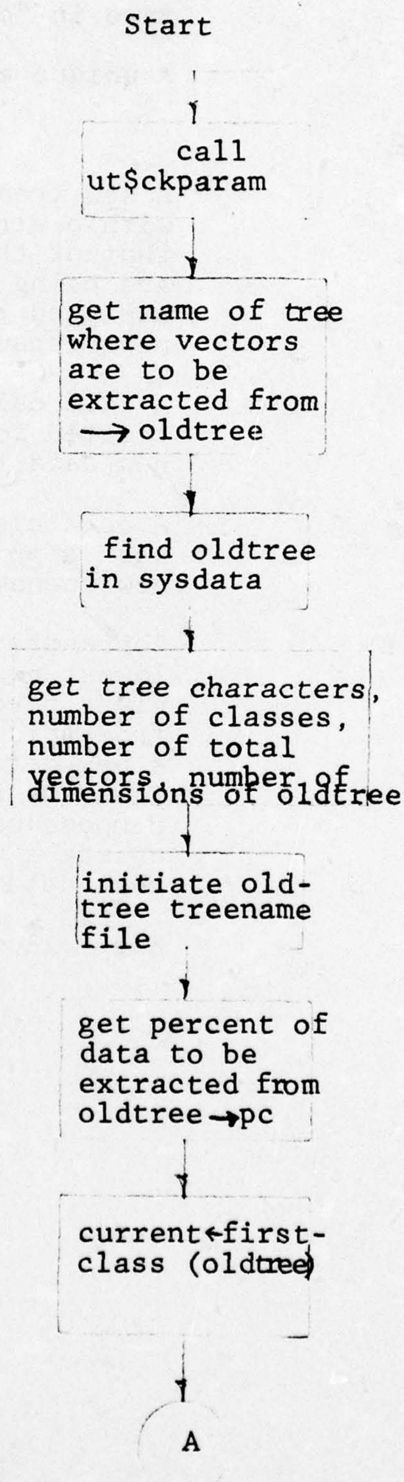

C
comb=1
or fid = yes?
no
yes
Create output
file
Call lnodes for each
tree, set up array
ld (i, j) i=1 to numtree,
j=1 to number of
lowest nodes
comb = 1?
yes
D
no
opt = 1?
yes
User interaction
decides scratch
file
no
Set up scratch so that
all similarly named
nodes are merged into
new node of same name
E

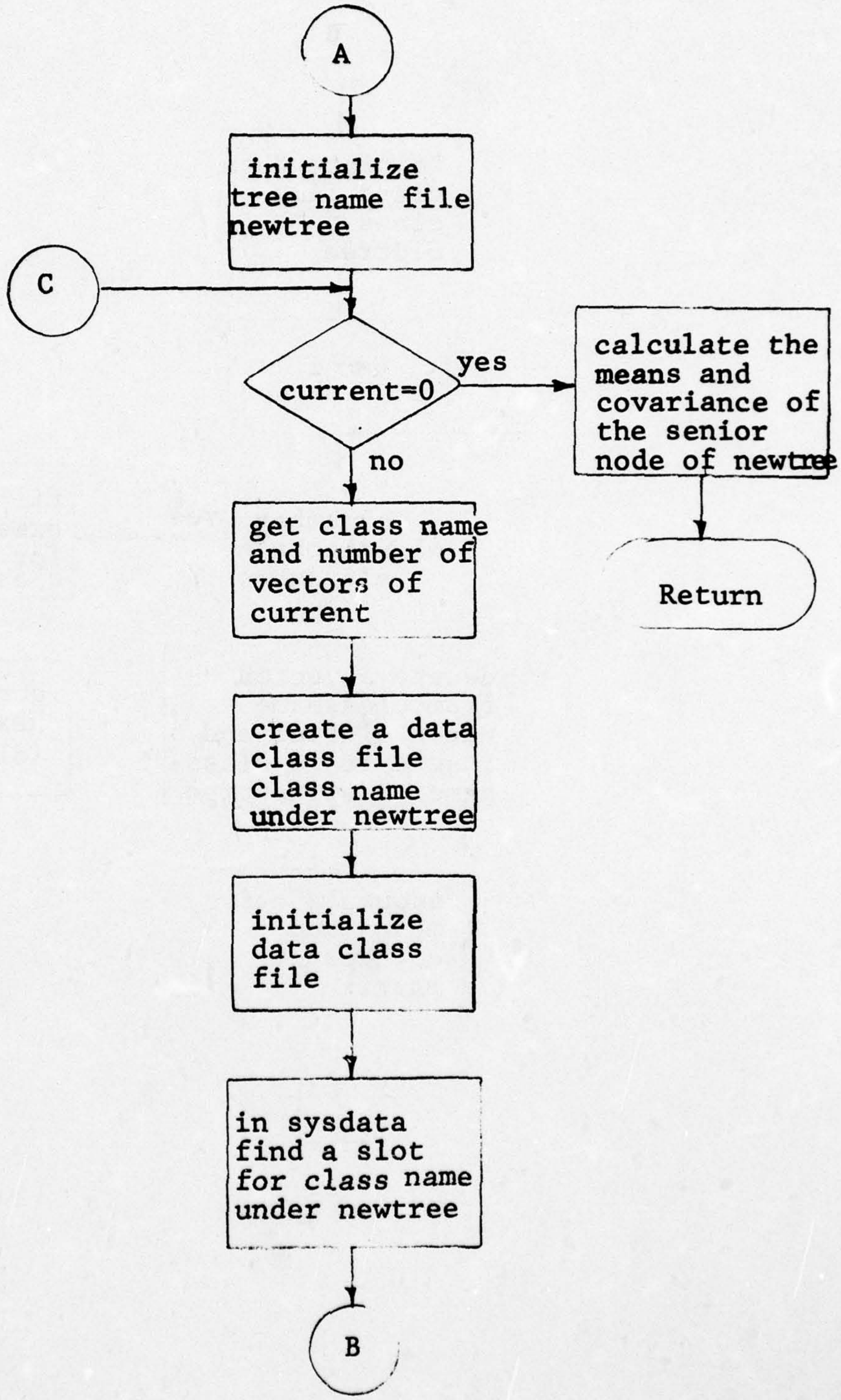

D
Set up scratch so that new node has same name as old node & is same as old node
comb = 2
Display char. unique
yes
no
Change chars., output changes
comb = 1
E
Copy vectors into new dataclass files
F

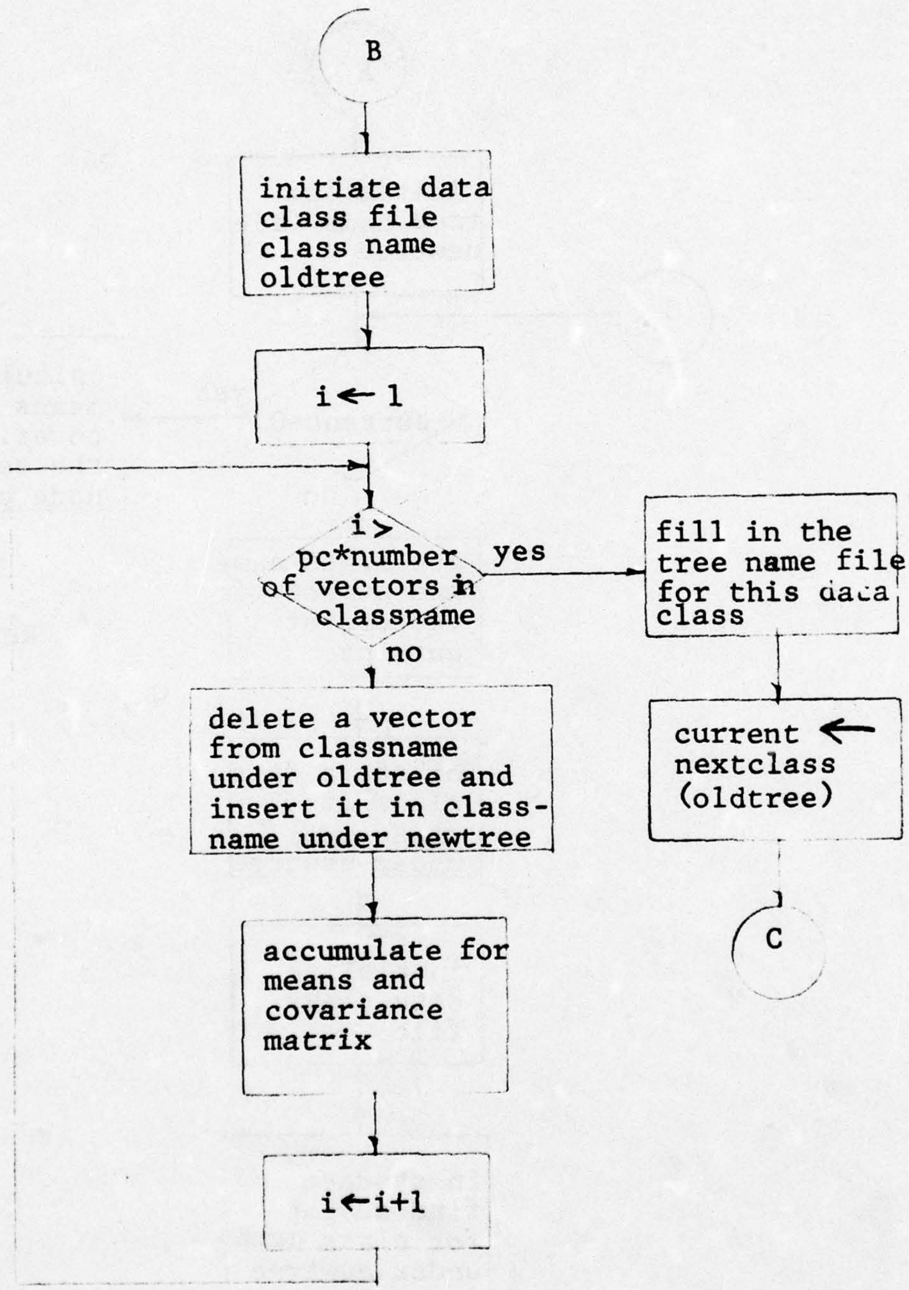

F
decis = yes?
yes
Sequence id's
no
Set up treename file
Set up sysdata
comb = 1 or fid = yes?
no
yes
Output file
G
Return

MOOS Function Name:	crrandts
Calling Sequence:	type in "crrandts(newtreename)"
Input Parameter:	A unique eight-character treename
Output File Settings:	
"sysdata"	A new tree is created in sysdata with a structure identical to that of the tree from which data are being extracted. The name assigned to this new tree is newtreename.
treename file	A file called "newtreename" is created for the vectors under the new data tree.
dataclass files	A data class file is created for each a priori data class under newtreename.
Program Description:	"crrandts," given a data tree of lowest nodes, randomly removes a given percentage of the vectors from this data tree and creates a new data tree which will contain the removed vectors. The purpose of this program is to create a work data set and a test data set.
Flow Chart:	See following pages

crrandts


Start
call ut$ckparam
get name of tree where vectors are to be extracted from → oldtree
find oldtree in sysdata
get tree characters, number of classes, number of total vectors, number of dimensions of oldtree
initiate old-tree treename file
get percent of data to be extracted from oldtree → pc
current ← first-class (oldtree)
A


A
initialize tree name file newtree
C
current=0
yes
no
calculate the means and covariance of the senior node of newtree
Return
get class name and number of vectors of current
create a data class file class name under newtree
initialize data class file
in sysdata find a slot for class name under newtree
B


B
initiate data
class file
class name
oldtree
i ← 1
i > pc*number of vectors in classname
yes
no
fill in the tree name file for this data class
current ← nextclass (oldtree)
C
delete a vector from classname under oldtree and insert it in classname under newtree
accumulate for means and covariance matrix
i ← i+1

Internal Subroutine Name: ctsm

Calling Sequence: call ctsm (ptrs, ln)

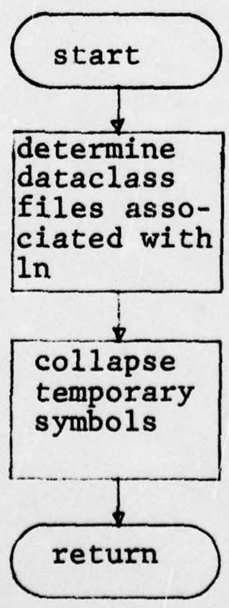
Input Parameters:

ptrs	(5)ptr	ptrs(1) - "sysdata" ptrs(4) - "treename" ptrs(5) - "mooslogic file"
ln	fixed (35) logic node whose temporary symbols are to be collapsed	

Output File Settings: The temporary symbols of any vectors in the dataclass files associated with (ln) are examined. If a given temporary symbol is from a node lower than (ln) in the logic tree, it is reset to ln.

Program Description: ctsm searches all dataclass files associated with logic node (ln) for vectors whose temporary symbols are equal to logic nodes below logic node (ln) in the logic tree. The temporary symbols are then set to ln.

Flow Chart:



MOOS Function Name: dataprnt

MOOS Function Number: 25

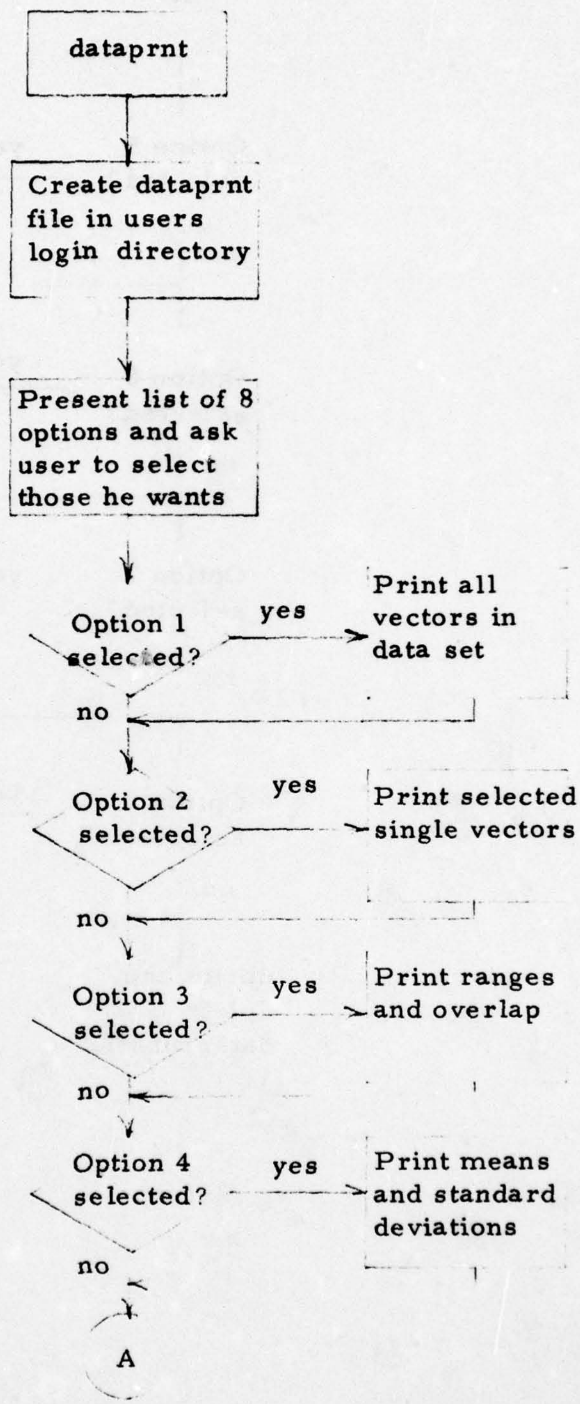
Calling Sequence: Type in "dataprnt [(treename)] [(nodename)]"

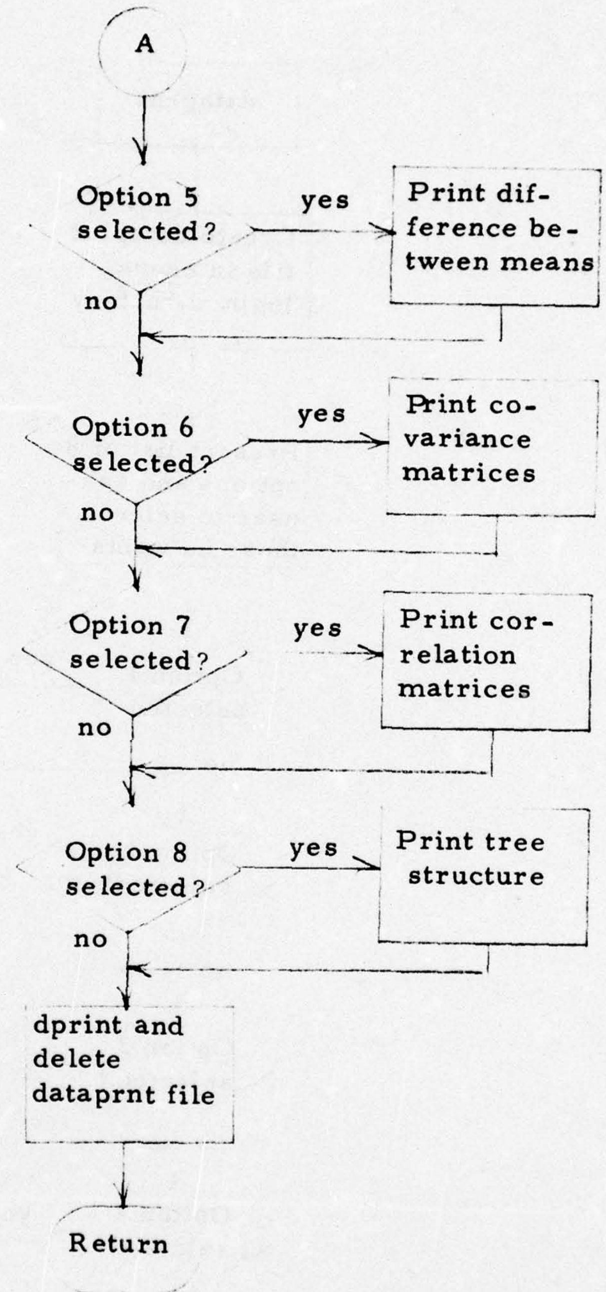
Input Parameters: Standard optional data set selection parameters

Output File Settings: dataprnt_file is created in user's login directory. All output from dataprnt is placed in this file.

Program Description: dataprnt consists of eight basic printout options which allow the user to get certain basic statistical information about a data set. Most of the routine is involved with formating information which already exists (such as mean vectors and covariance matrices which are stored in the treename file of the data set. If the "e" option is selected, values are printed in an exponential format. The "c" option allows the user to designate a subset of the selected data set to be processed. More detailed explanation of the available options can be found in the user documentation.

Flow Chart: See following page.


dataprnt
Create dataprnt file in users login directory
Present list of 8 options and ask user to select those he wants
Option 1 selected?
yes
Print all vectors in data set
no
Option 2 selected?
yes
Print selected single vectors
no
Option 3 selected?
yes
Print ranges and overlap
no
Option 4 selected?
yes
Print means and standard deviations
no
A


A
Option 5 selected?
yes
Print difference between means
no
Option 6 selected?
yes
Print covariance matrices
no
Option 7 selected?
yes
Print correlation matrices
no
Option 8 selected?
yes
Print tree structure
no
dprint and delete dataprnt file
Return

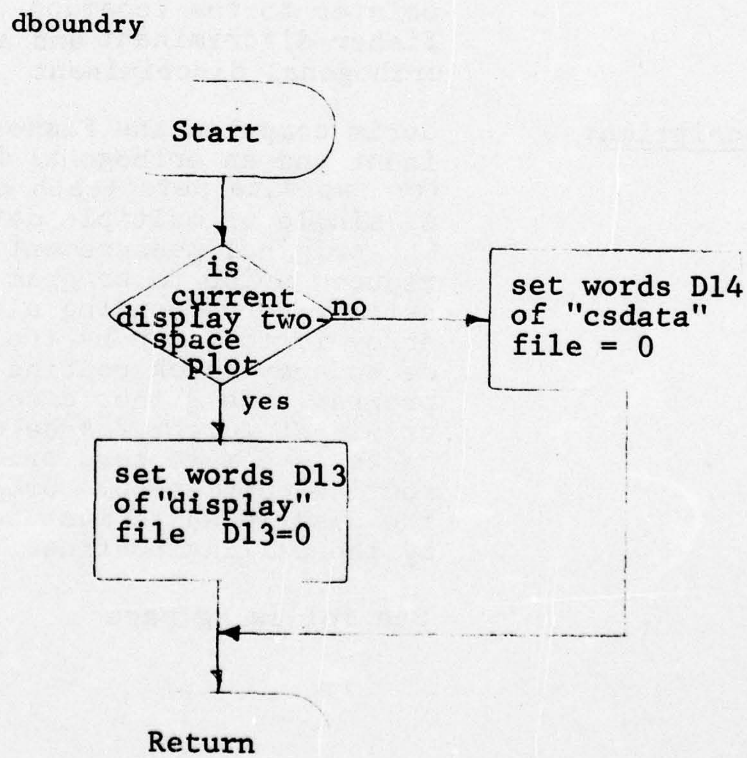
Utility Function Name: dboundry

Calling Sequence: type in "dboundry"

Output Files Settings: The words D13 of the "display" file, if current display is a two-space plot or words D14 of "csdata", if current display is a histogram are set to zero.

Program Description: The system display code is examined. If it is 1 the words D13 of the display file are set to 0, if it is a 4 then words D14 of the "csdata" file are initiated to 0. Then the appropriate display program, with "clusscat" or "npcos" is called to present the display.

Flow Chart:



Internal Subroutine Name: dcrim

Calling Sequence: **call dcrim (ptrd, ptrn, ptric,** ptrs, ndim)

Input Parameters:

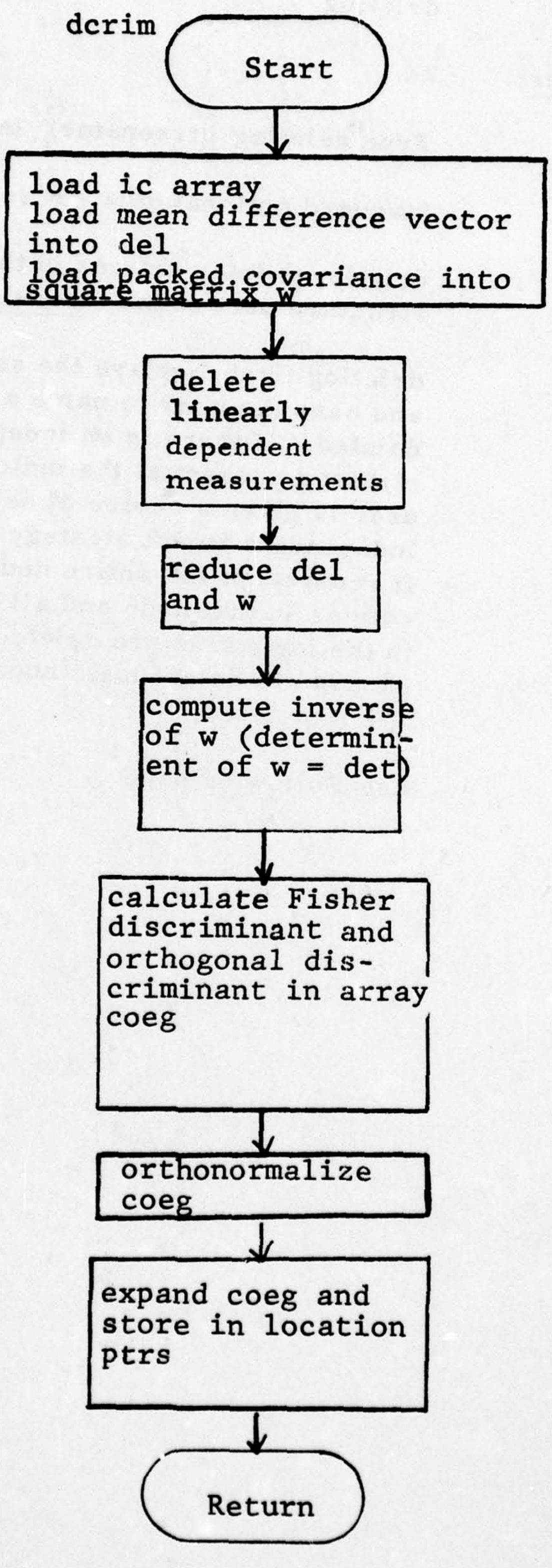
ptrd	-	pointer to a mean difference vector
ptrw	-	pointer to the packed lumped covariance matrix
ptric	-	pointer to an array of ndim words set to 0 (include measurement) or 1 (do not include measurement)
ptrs	-	pointer to a storage array for the Fisher discriminant and an orthogonal discriminant (must be at least 2*ndim words in length)
ndim	-	dimensionality of the data

Output Parameter:

ptrs	-	pointer to the location of the stored Fisher discriminant and an optional orthogonal discriminant

Program Description: dcrim computes the Fisher discriminant and an orthogonal discriminant for two data sets (each composed of single or multiple data classes.) The original measurement set may be reduced prior to program call (by setting corresponding elements in array ic to 1) or by the linear dependency check routine within the program. In either case, the discriminant weight for deleted measurements is set to zero prior to subroutine completion. Storage area for the discriminants must be provided by the calling routine.

Flow Chart: **See following page**


dcrim
Start
load ic array
load mean difference vector into del
load packed covariance into square matrix w
delete linearly dependent measurements
reduce del and w
compute inverse of w (determinent of w = det)
calculate Fisher discriminant and orthogonal discriminant in array coeg
orthonormalize coeg
expand coeg and store in location ptrs
Return

MOOS Function Name: deletlog

MOOS Function Number: 28

Calling Sequence: Type"deletlog [(treename)] [(nodename)]"

Input Parameters: Standard optional data set selection parameters

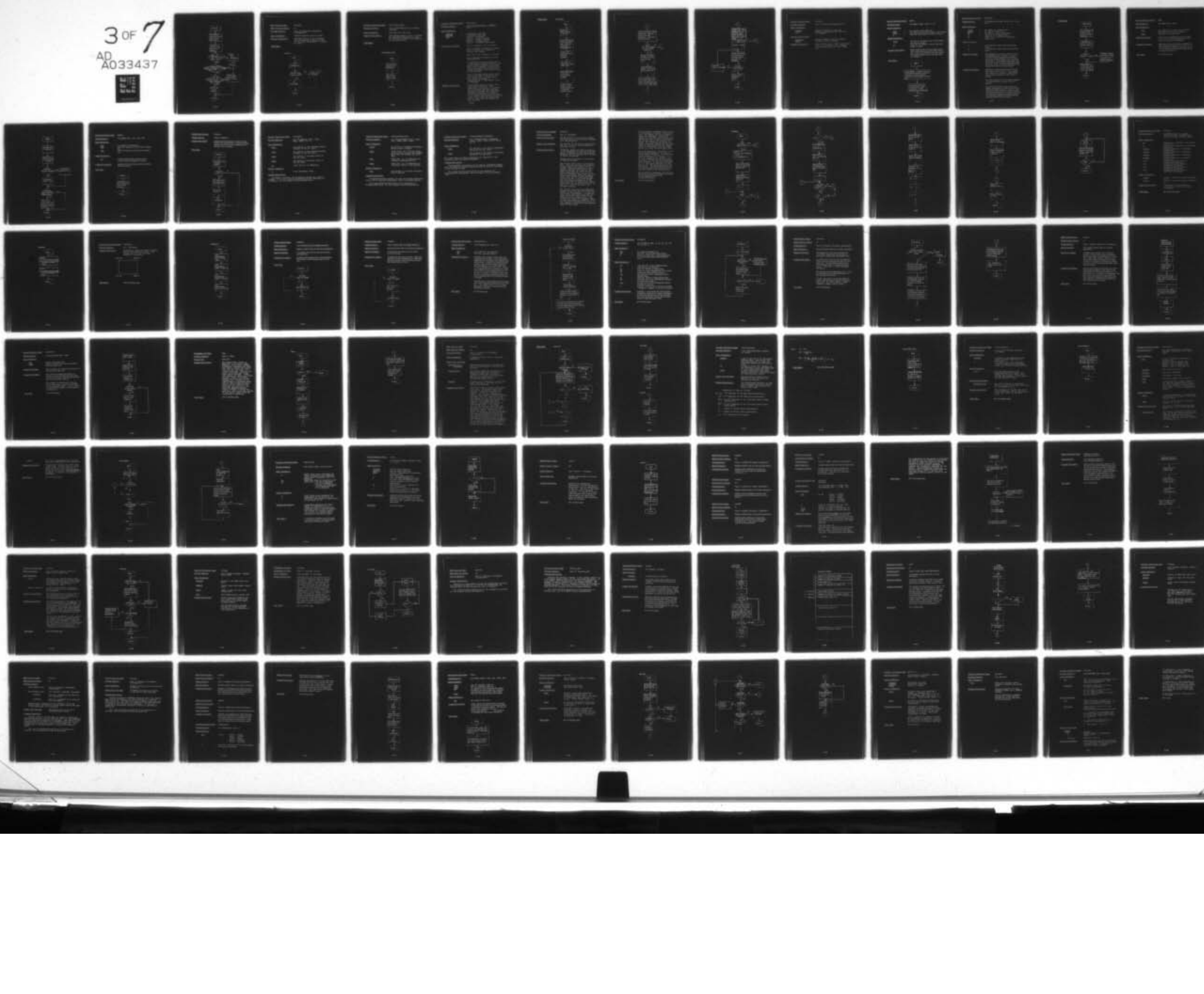
Output File Settings: deletlog deletes entries in the node part of the structure part of the "mooslogic file".

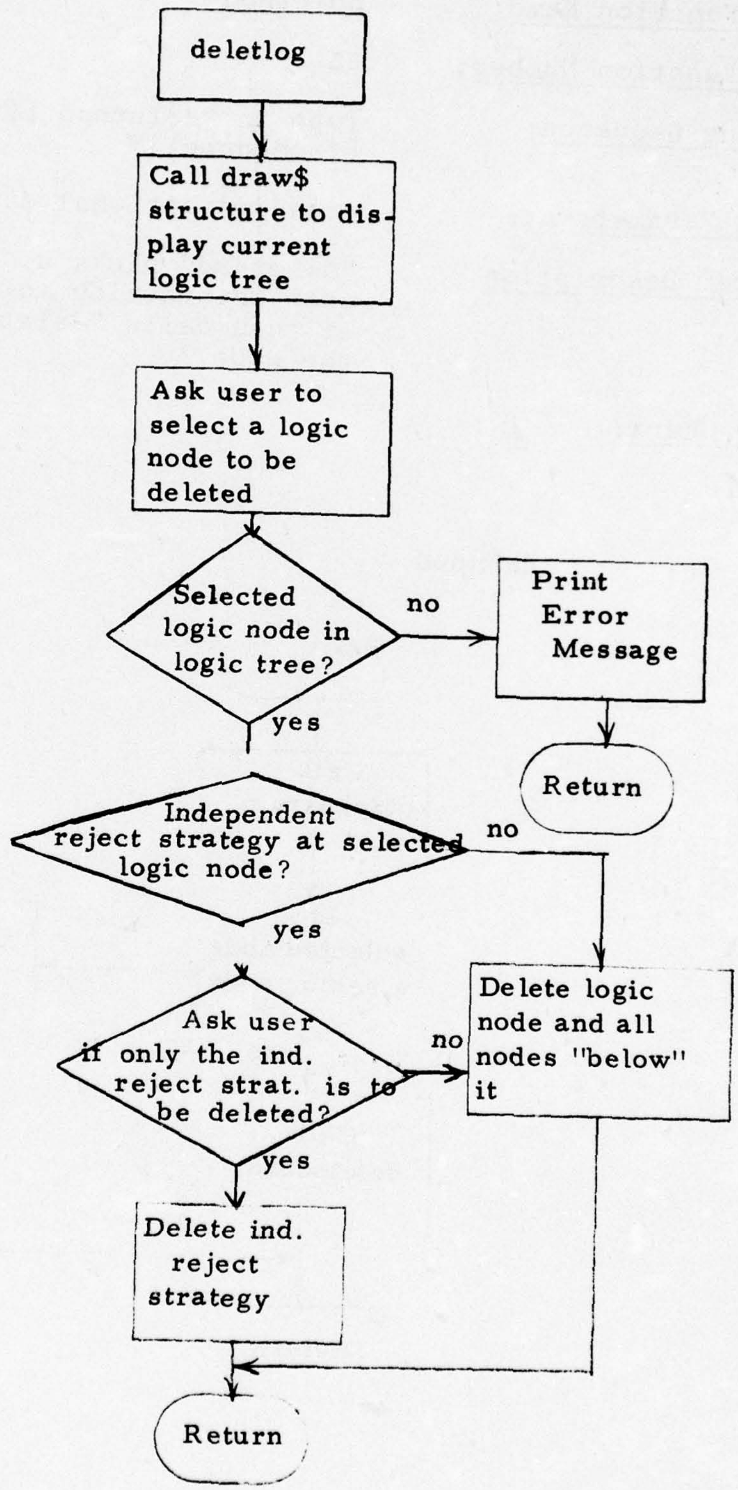
Program Description: deletlog first displays the selected logic tree and asks the user to name a logic node to be deleted. If there is an independent reject strategy present at the indicated node, the user is given a choice of deleting only the independent reject strategy or the entire node. If removal of the entire node is chosen, entries in that node and all nodes "below" it in the logic tree are deleted. If the user chooses to delete logic node 1 the mooslogic file is deleted.

Flow Chart: See Following page.

AD-A033 437 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
MULTICS OLPARS OPERATING SYSTEM.(U)
SEP 76 D B CONNELL, K N KLINGBAIL F30602-75-C-0226
UNCLASSIFIED PAR-74-25-B RADC-TR-76-271-VOL-2 NL

3 OF 7
AD A033437




deletlog
Call draw$ structure to display current logic tree
Ask user to select a logic node to be deleted
Selected logic node in logic tree?
no
Print Error Message
Return
yes
Independent reject strategy at selected logic node?
no
yes
Ask user if only the ind. reject strat. is to be deleted?
no
Delete logic node and all nodes "below" it
yes
Delete ind. reject strategy
Return

MOOS Function Name: deletnod

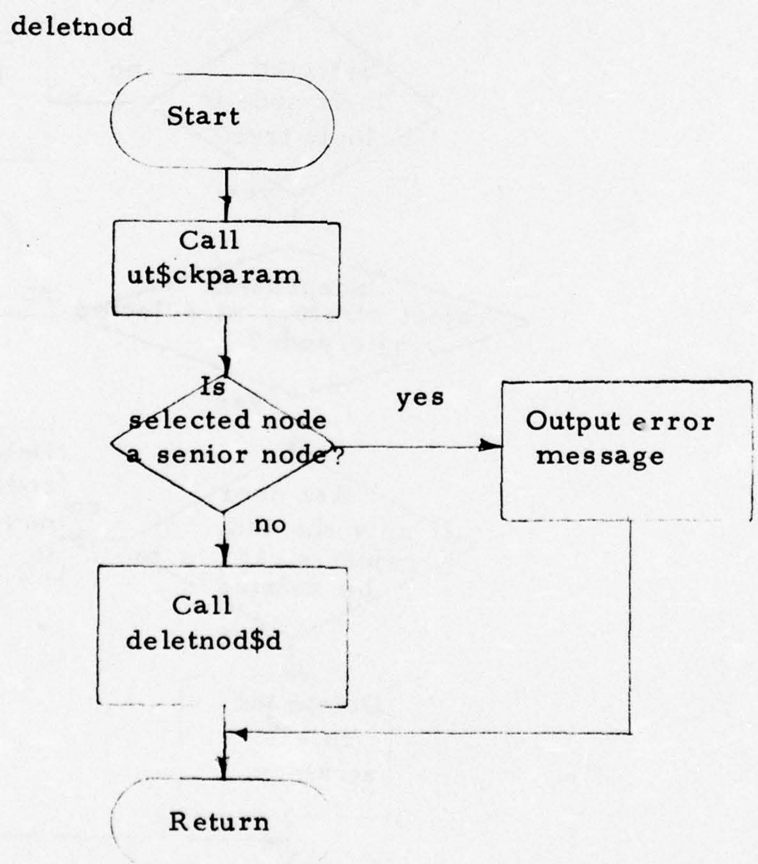
MOOS Function Number: 50

Calling Sequence: Type in "deletnod [(treename)] [(nodename)]"

Input Parameters: standard optional data set names

Program Description: "deletnod" picks up the necessary user interaction to delete a node. It then calls "deletnod$d" to delete the node.

Flow Chart:



Internal Subroutine Name: deletnod$ch_depth

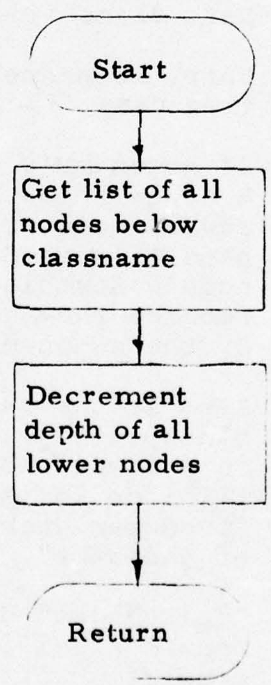
Calling Sequence: call deletnod$ch_depth (treename, nodename)

Input Parameters: tree name and class name

Program Description: "deletnod$ch_depth", given a treename and nodename, decrements by 1 the S6 entry of all nodes below nodename in the sysdata file.

Flow Chart:

deletnod$ch_depth



Internal Subroutine Name: deletnod$d

Calling Sequence: **call deletnod$d (treename, nodename, ptrs)**

Input Parameters:

treename	-	8-character tree name
nodename	-	4-character node name
ptrs	-	array of 4 pointers ptrs(1) = sysdata pointer ptrs(2) = scratch pointer ptrs(3) = display pointer ptrs(4) = treename file pointer

Output File Settings: css2 of sysdata is set to "nono".

Entry nodename is deleted in sysdata and all entries are changed to reflect this change.

Data class file nodename is deleted.

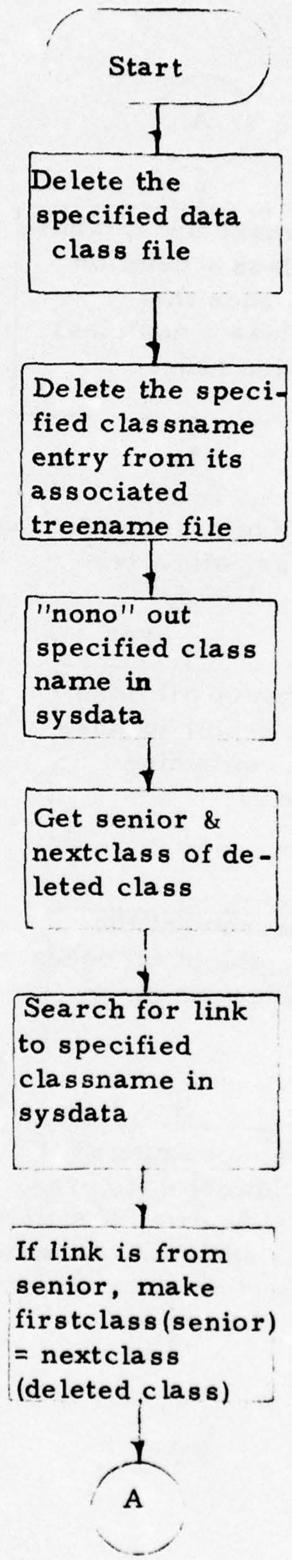
Entry nodename is deleted from the tree name file.

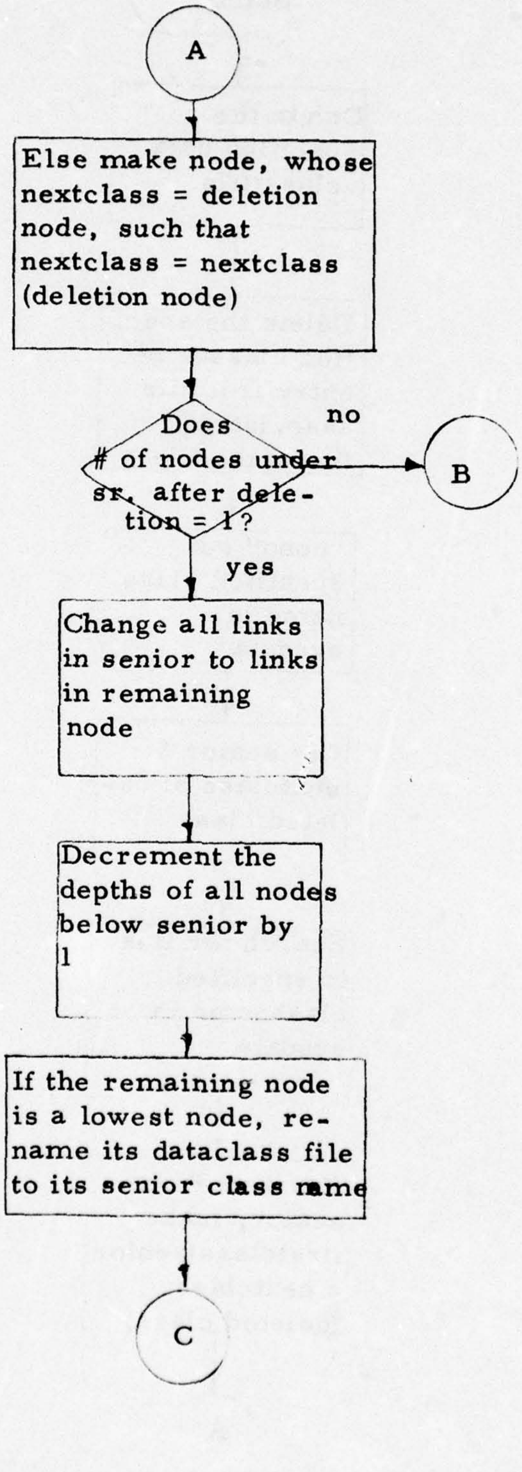
If nodename's senior node has only 2 nodes below it before deletion, the senior node of nodename in the tree name file is deleted, and the other node's associated data class file is renamed the name of the senior class of the nodename.

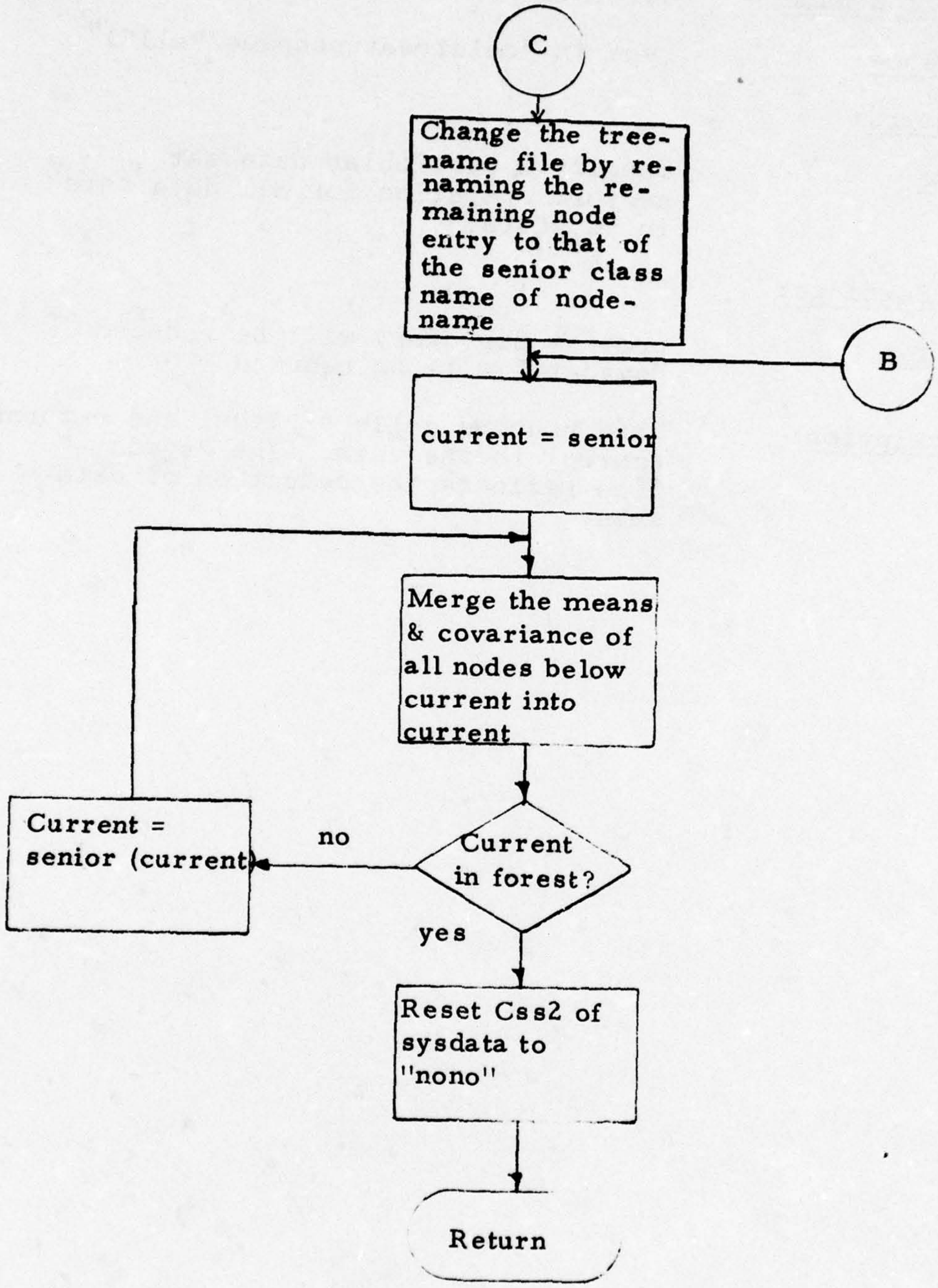
Also if the same situation of only 2 nodes exists and if the other node is a lowest node, the data class file for it is renamed to "treecharacter" || "the senior class of nodename".

Program Description: "deletnod$d" deletes a lowest node from a specified tree. Upon completion of deleting, it traverses back up through the data tree and recomputes the means and covariance matrices to reflect the deletion. If the specified node to delete is one of the two nodes below the deleted node's senior node, the senior node and the other node are combined to form one node.

Flow Chart: deletnod$d




A
Else make node, whose nextclass = deletion node, such that nextclass = nextclass (deletion node)
Does # of nodes under sr. after deletion = 1?
no
B
yes
Change all links in senior to links in remaining node
Decrement the depths of all nodes below senior by 1
If the remaining node is a lowest node, rename its dataclass file to its senior class name
C


C
Change the tree-name file by renaming the remaining node entry to that of the senior class name of node-name
B
current = senior
Merge the means & covariance of all nodes below current into current
Current in forest?
no
Current = senior (current)
yes
Reset Css2 of sysdata to "nono"
Return

Utility Function Name: deletree

Calling Sequence: type in "deletree(treename/"all")"

Input Parameters:

treename	specify a particular data set
"all"	perform operation for all data sets in "sysdata."

Output File Settings:

Directory	process directory will be reduced
Tables	"sysdata" will be reduced

Program Description: This routine calls s_p$tdel and returns control to the user. The "sysdata" file reflects the reduction of data sets.

Internal Subroutine Name: dg$acl

Calling Sequence: call dg$acl (trptr, array, n, m)

Input Parameters:

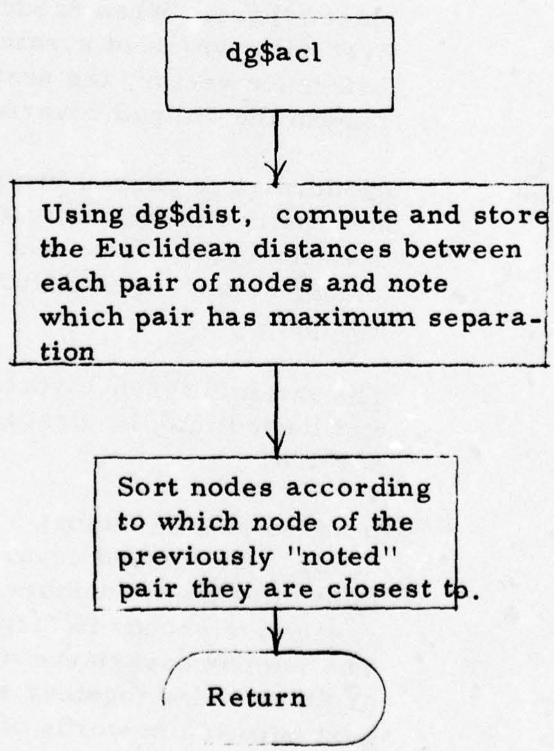
trptr	-	ptr pointer to tree name file.
array	-	(72) char (4) array of node names to be stored.
n	-	fixed (35) total number of nodes in (array).

Output Parameters:

array	-	(72) char (4) the given array of node names, sorted into 2 groups.
m	-	fixed (35) the number of nodes in the first group in (array).

Program Description: dg$acl finds the pair of nodes whose mean vectors have the greatest Euclidean separation. The rest of the nodes are then sorted according to which nodes of this pair they lie closest to.

Flow Chart:



Internal Subroutine Name: dg$dcrmsu

Calling Sequence: call dg$dcrmsu (treeptr, ptrscr, ar, n, m, a, b)

Input Parameters:

treeptr	-	ptr pointer to "treename" file.
ptrscr	-	ptr point to "scratch" file
ar	-	(72) char (4) array of node names.
n	-	fixed (35) no. of names in (ar).
m	-	fixed (35) no. of nodes in first group of nodes in (ar).

Output Parameters:

a	-	(100) fixed (35) mean vector for the first group.
b	-	(100) fixed (35) mean vector for the second group.

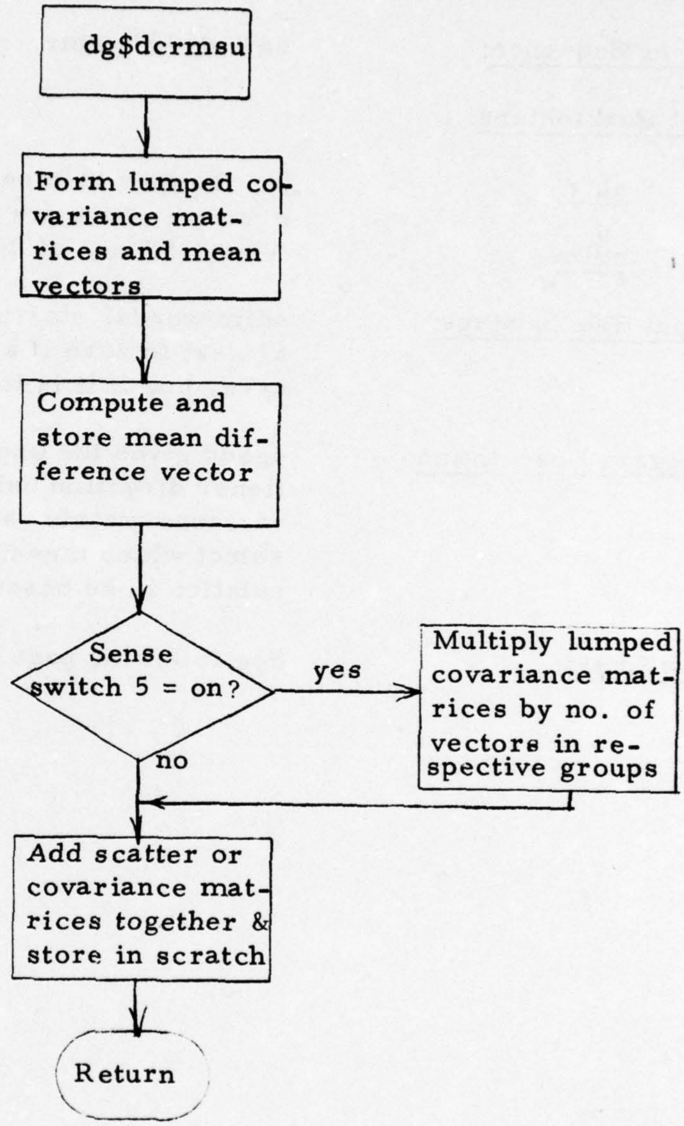
Output File Settings: The scratch file is used as a buffer area for merging means and covariance matrices. The final lumped and unpacked covariance matrix and mean difference vector are then placed there. When dg$dcrmsu returns, the first ndim words of scratch contain the mean difference vector, the next ndim*ndim words contain the lumped covariance matrix.

Program Description: dg$dcrmsu is passed two groups of lowest node names in the array ar. The program uses mmeanacv to produce a lumped mean and covariance matrix for each of these groups.

The mean difference vector is then computed and loaded into the first ndim words of scratch.

If sense switch number 5 has been set (by dg$dd), the lumped covariance matrices are multiplied by the number of vectors in their respective groups to form "scatter" matrices. The lumped covariance or scatter matrices are then added together and loaded into the next ndim*ndim words of scratch.

Flow Chart:


dg$dcrmsu
Form lumped covariance matrices and mean vectors
Compute and store mean difference vector
Sense switch 5 = on?
yes
no
Multiply lumped covariance matrices by no. of vectors in respective groups
Add scatter or covariance matrices together & store in scratch
Return

Internal Subroutine Name: dg$dd

Calling Sequence: call dg$dd (sptr, ndim)

Input Parameters:

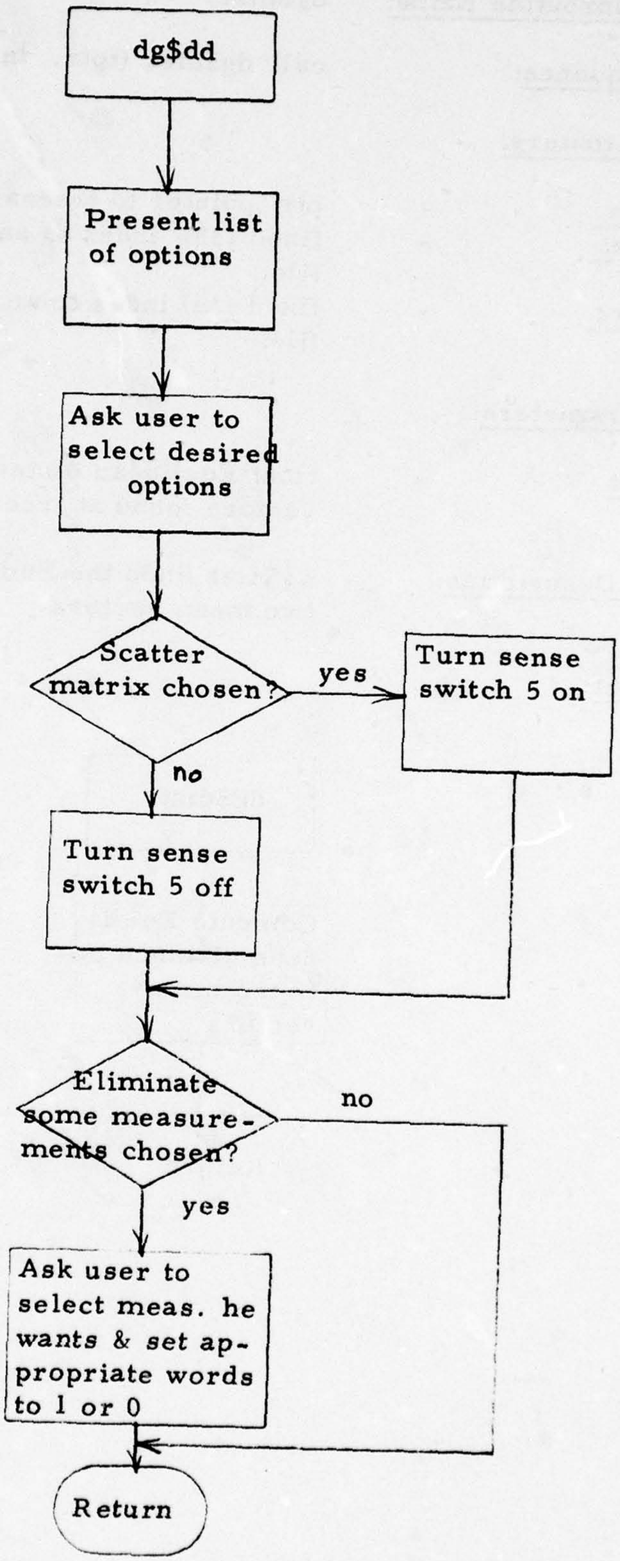
sptr - ptr pointer to area which indicates which measurements are to be eliminated.

ndim - fixed (35) no. of dimensions.

Output File Settings: ndim words, starting at the location of sptr, are set to zero if a measurement is to be used, one if it is to be eliminated.

Program Description: dg$dd gives the user the choice of finding the fisher direction using either a scatter or covariance matrix, and also allows the user to select which measurements he wishes the calculation to be based on.

Flow Chart: See following page.


dg$dd
Present list of options
Ask user to select desired options
Scatter matrix chosen?
yes
Turn sense switch 5 on
no
Turn sense switch 5 off
Eliminate some measure-ments chosen?
no
yes
Ask user to select meas. he wants & set ap-propriate words to 1 or 0
Return

Internal Subroutine Name: dg$dist

Calling Sequence: call dg$dist (tptr, inxl, inx2, dst)

Input Parameters:

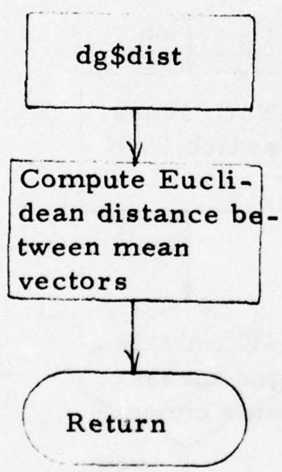
tptr	-	ptr pointer to treename file.
inxl	-	fixed (35) index to an entry in the treename file.
inx2	-	fixed (35) index to an entry in the treename file.

Output Parameters:

dst	-	float Euclidean distance between mean vectors found at treename file entries.

Program Description: dg$dist finds the Euclidean distance between two mean vectors

Flow Chart:

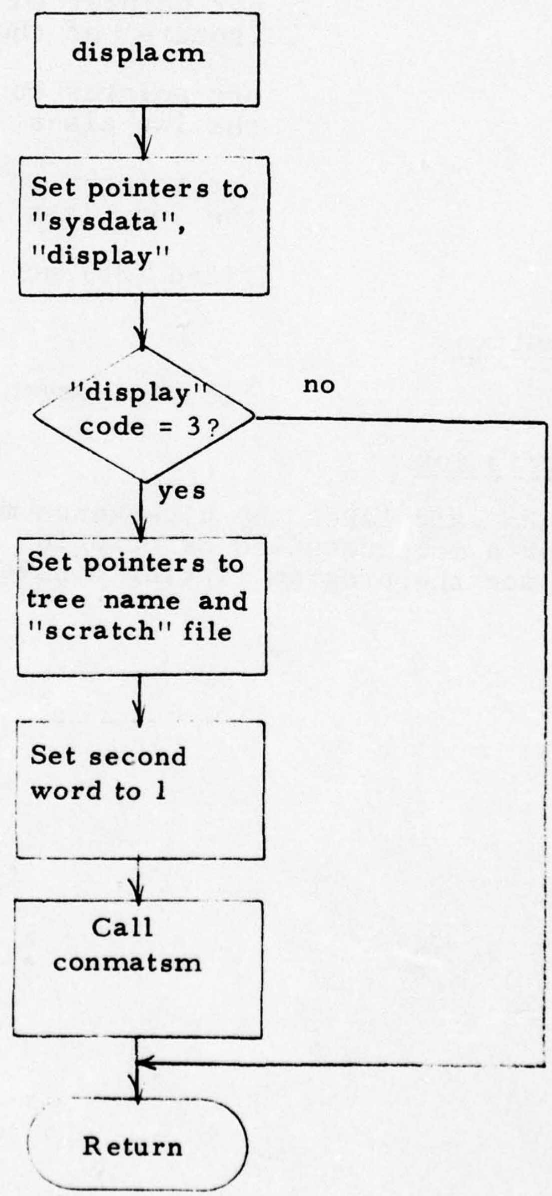


Utility Function Name: displacm

Calling Sequence: Type in "displacm"

Program Description: displacm generates the four pointers if the display file code indicates a confusion matrix, sets the second word of the display file to 1, and then calls conmatsm.

Flow Chart:



Internal Subroutine Name: divergence

Calling Sequence: call divergence (clptr, c2ptr, mlptr, m2ptr, nd, J)

Input Parameters:

clptr	ptr pointer to the covariance matrix (square) of the 1st class
c2ptr	ptr pointer to the covariance matrix (square) of the 2nd class
mlptr	ptr pointer to the mean vector of the 1st class
m2ptr	ptr pointer to the mean vector of the 2nd class
nd	fixed (35) no. of dimensions

Output Parameters:

J	float divergence value

Program Description:

divergence calculates the divergence measure for a pair of classes. For a more detailed description of the operation of divergence, see the program listing documentation.

Internal Subroutine Name: divergence$covsetup

Calling Sequence: call divergence$covsetup (covptr, cptr, smeas, ndim, order)

Input Parameters:

covptr — ptr pointer to original covariance matrix (lower triangle)

smeas — (100) fixed (35) non-zero values in this array indicate measurements whose rows and columns are to be kept

ndim — fixed (35) no. of dimensions of original covariance matrix

order — fixed (35) no. of dimensions of covariance matrix to be returned

Output Parameters:

cptr — ptr pointer to returned covariance matrix (square)

Program Description:

divergence$covsetup removes the rows and columns associated with a set of selected measurements from a covariance matrix.

For a more detailed description of the operation of divergence$covsetup, see the program listing documentation.

Internal Subroutine Name: divergence$fast_divergence

Calling Sequence: call divergence$fast_divergence (c1ptr, c2ptr, i1ptr, i2ptr, m1ptr, nd, J)

Input Parameters:

i1ptr — ptr pointer to the inverse covariance matrix of the 1st class

i2ptr — ptr pointer to the inverse covariance matrix of the 2nd class

All other input and output parameters are identical to the parameters used by program divergence.

Program Description:

divergence$fast_divergence is the same as divergence except that the required inverse covariance matrices must be calculated by the calling routine.

For a more detailed description of the operation of divergence$fast_divergence, see the program listing documentation.

Utility Function Name: dra$bndy

Calling Sequence: Type in "dra$bndy"

Input File Settings: The D8 portion of the cluster display file must be set if the current display is a two-space plot.

Output File Settings: D13 and D14 of the cluster display file are set, or D14 of the histogram display file is set.

Program Description: "dra$bndy" allows the user to draw up to two boundaries or thresholds in two-space (a maximum of five line segments/boundary) on the tektronix after a plot is put on the screen.

Drawing these boundaries in two-space is done as follows:

The routine first turns on the crosshair. The user may move the crosshair to any point on the screen that he wishes. When the crosshair is in the desired location and he wishes that point to be read, he enters one of three characters ["c," "e," or "q"]. The "c" (continue) means that the user wishes more points for this boundary to be read; the "e" (end) means that this is the end of the first boundary but that another boundary is to be drawn; and the "q" (quit) means that no more points for any boundary are to be read (i.e. this is the end of all boundary-drawing).

The routine reads the first character entered along with the tektronix coordinates of the crosshair. It then converts these coordinates to projection plane coordinates and stores the information in the display file. The routine then turns the crosshair back on and this process is repeated. After the tektronix coordinates are sent, a line segment is drawn from the previous point to this point.

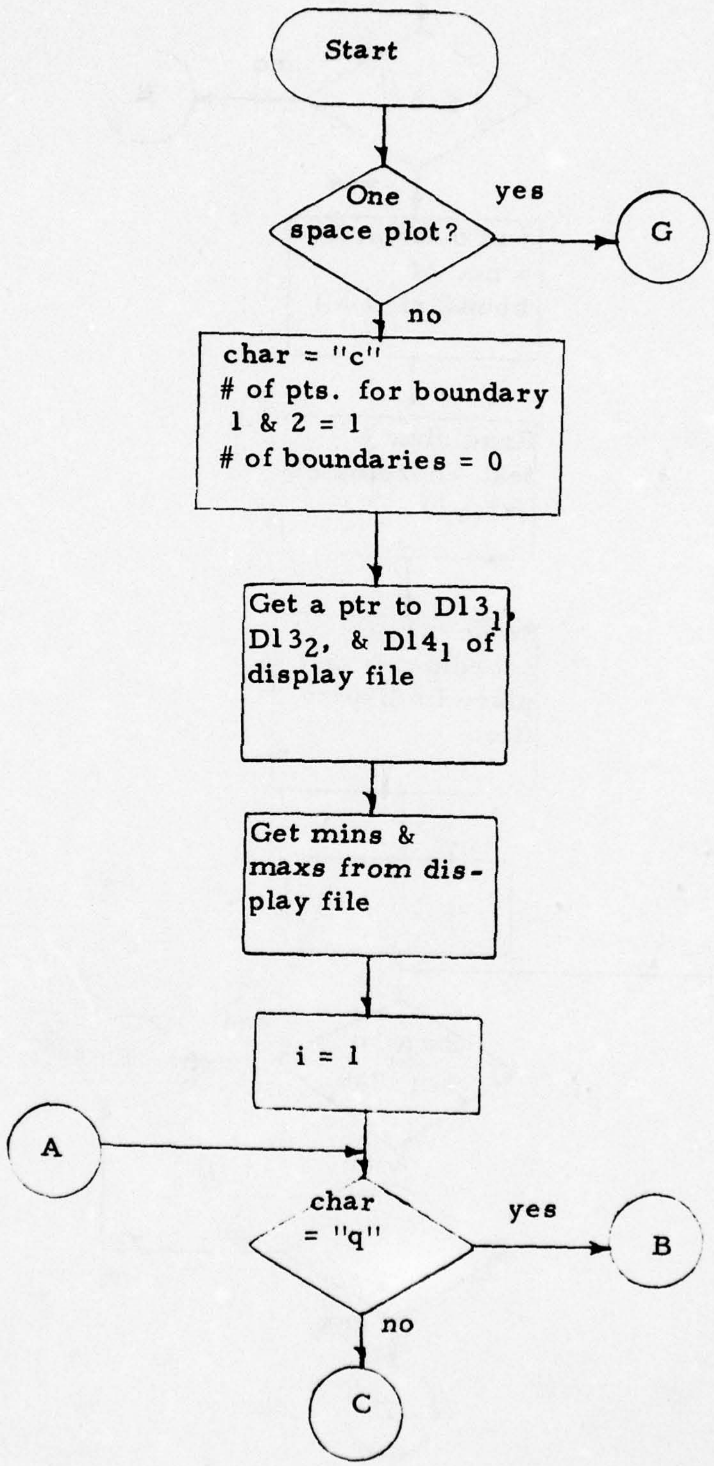
This process is repeated until the end of the boundary is drawn. After the last line segment for each boundary is drawn, the crosshair (x-hair) is turned on once again; the routine now expects the user to move the x-hair to the convex side of the boundary just drawn. Any character can now be entered; the point occupied by this character is read as being on the convex side of the boundary, and this information is stored in the display file.

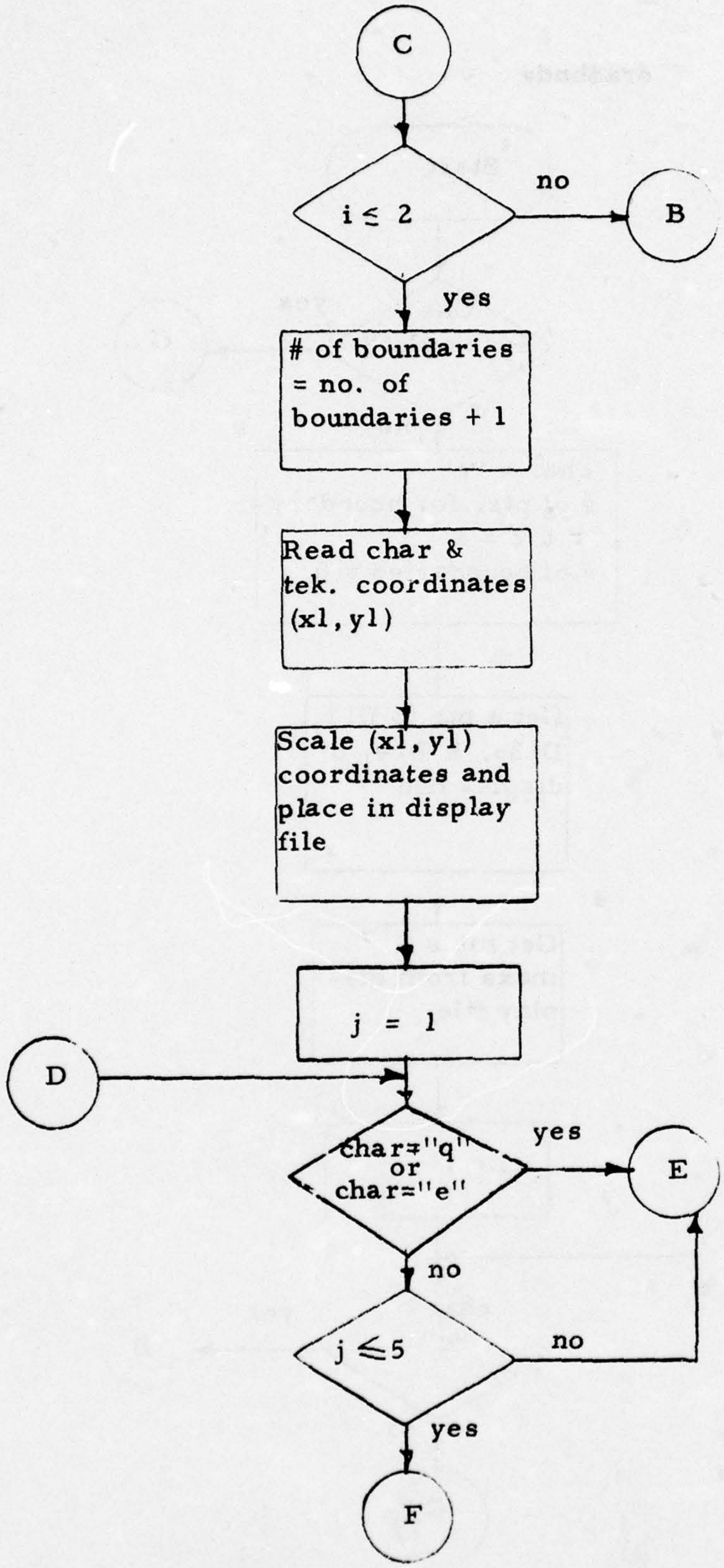
In a one-space plot, the x-hair is enabled and positioned to the spot of the intended threshold. The user then enters either a "q" or an "e." If only one threshold is desired, a "q" is hit; otherwise the crosshair will be turned on twice and two thresholds must be drawn. If the first character is "e," the second threshold is sent with a "q." No convex points need to be entered. The tektronix points entered are converted to threshold values and stored in the display file.

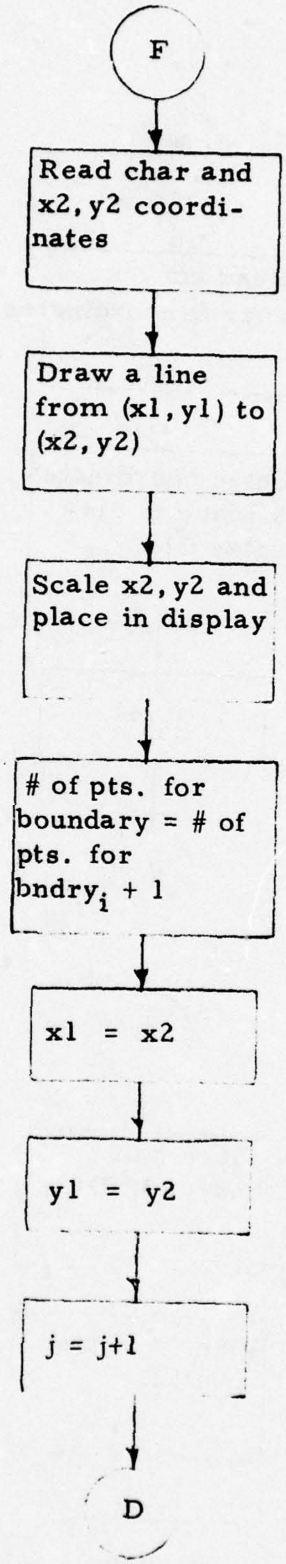
All tektronix coordiantes for both one-space and two-space displays are read by the internal subroutine <u>multeks$read_xhair</u>.

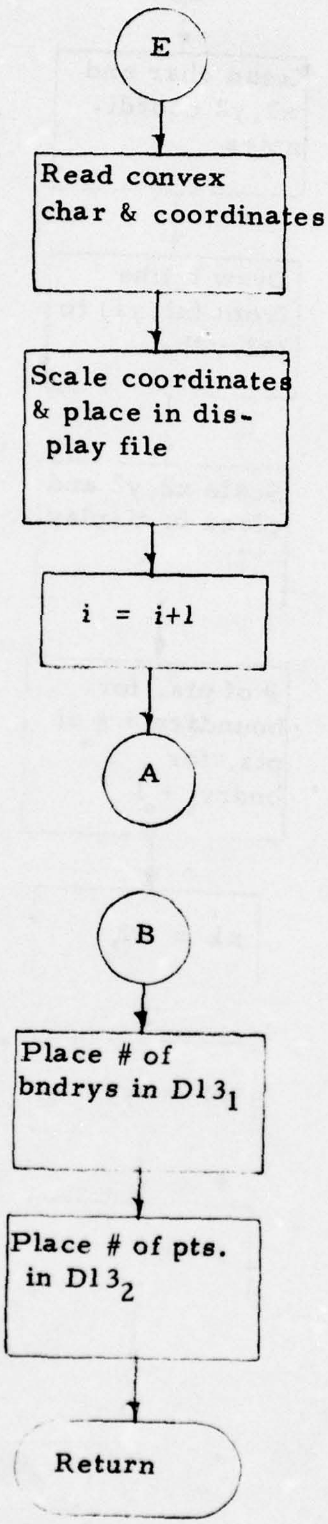
<u>Flow Chart</u>: See following page

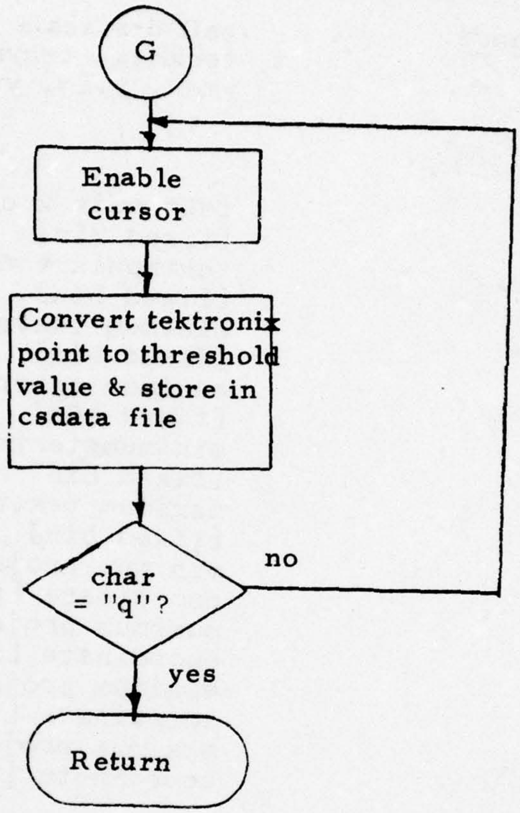
dra$bndy


Start
One space plot?
yes
G
no
char = "c"
# of pts. for boundary 1 & 2 = 1
# of boundaries = 0
Get a ptr to $D13_1$, $D13_2$, & $D14_1$ of display file
Get mins & maxs from display file
i = 1
A
char = "q"
yes
B
no
C


C
i ≤ 2
no
B
yes
# of boundaries = no. of boundaries + 1
Read char & tek. coordinates (xl, yl)
Scale (xl, yl) coordinates and place in display file
j = 1
D
char="q" or char="e"
yes
E
no
j ≤ 5
no
yes
F


F
Read char and x2, y2 coordinates
Draw a line from (x1, y1) to (x2, y2)
Scale x2, y2 and place in display
# of pts. for boundary = # of pts. for bndry$_i$ + 1
x1 = x2
y1 = y2
j = j+1
D


E
Read convex char & coordinates
Scale coordinates & place in display file
i = i+1
A
B
Place # of bndrys in $D13_1$
Place # of pts. in $D13_2$
Return


G
Enable cursor
Convert tektronix point to threshold value & store in csdata file
char = "q"?
no
yes
Return

Internal Subroutine Name: dra$scale

Calling Sequence: **call dra$scale (xl, yl, tecxmin,** tecxmax, tecymin, tecymax, ymin, ymax, ymin, ymax, xscale, yscale)

Input Parameters:

xl	tektronix x coordinate to be scaled [fixed bin]
yl	tektronix y coordinate to be scaled [fixed bin]
tecxmin	minimum tektronix x coordinate [fixed bin]
tecxmax	maximum tektronix x coordinate [fixed bin]
tecymin	minimum tektronix y coordinate [fixed bin]
tecymax	maximum tektronix y coordinate [fixed bin]
xmin	minimum projection plane x coordinate [floating]
xmax	maximum projection plane x coordinate [floating]
ymin	minimum projection plane y coordinate [floating]
ymax	maximum projection plane y coordinate [floating]

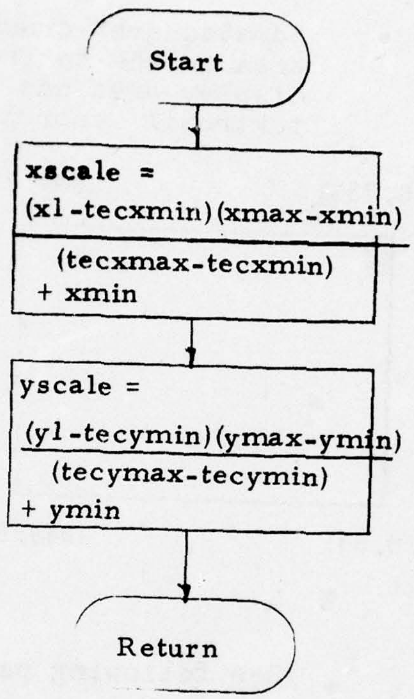
Output Parameters:

xscale	scaled x projection plane coordinate of xl
yscale	scaled y projection plane coordinate yl

Program Description: "dra$scale" scales tektronix coordinates to projection plane coordinates

Flow Chart: **See following page**

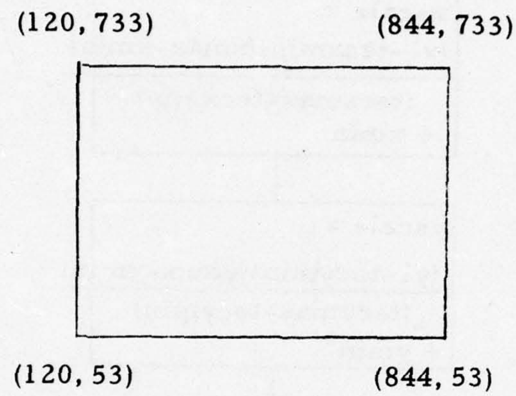
dra$scale


Start
xscale =
(xl-tecxmin)(xmax-xmin)
(tecxmax-tecxmin)
+ xmin
yscale =
(yl-tecymin)(ymax-ymin)
(tecymax-tecymin)
+ ymin
Return

Internal Subroutine Name: dra$square

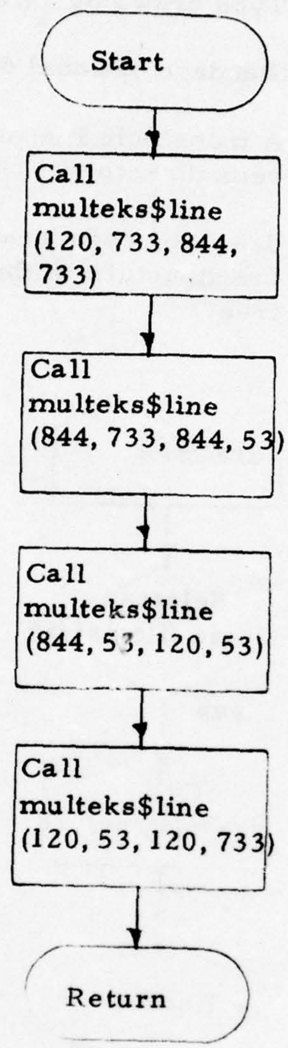
Calling Sequence: call dra$square

Program Description: "dra$square" draws the display viewing area on the tektronix screen. This display area has the following tektronix coordinates.



Flow Chart: See following page

dra$square


Start
Call multeks$line (120, 733, 844, 733)
Call multeks$line (844, 733, 844, 53)
Call multeks$line (844, 53, 120, 53)
Call multeks$line (120, 53, 120, 733)
Return

Utility Function Name: draw$log

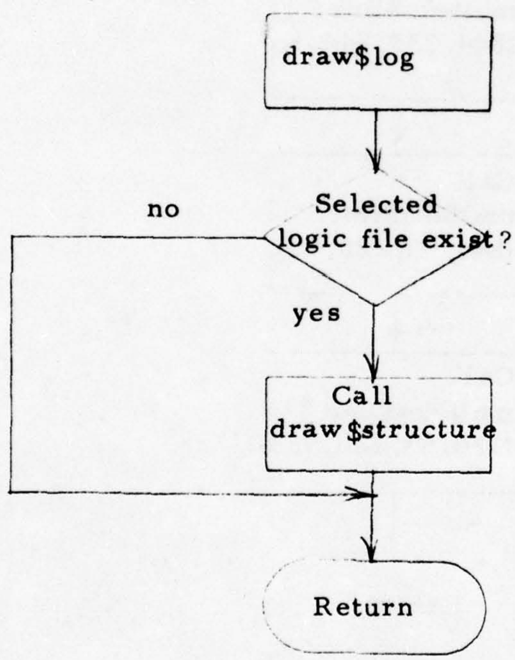
Calling Sequence: Type "draw$log [(treename)][(nodename)] "

Input Parameters: Standard optional data set selection parameters

Input File Settings: A mooslogic file must be present in the process directory

Program Description: draw$log calls draw$structure which displays the structure of the selected logic file as a tree.

Flow Chart:



Utility Function Name: draw$prt

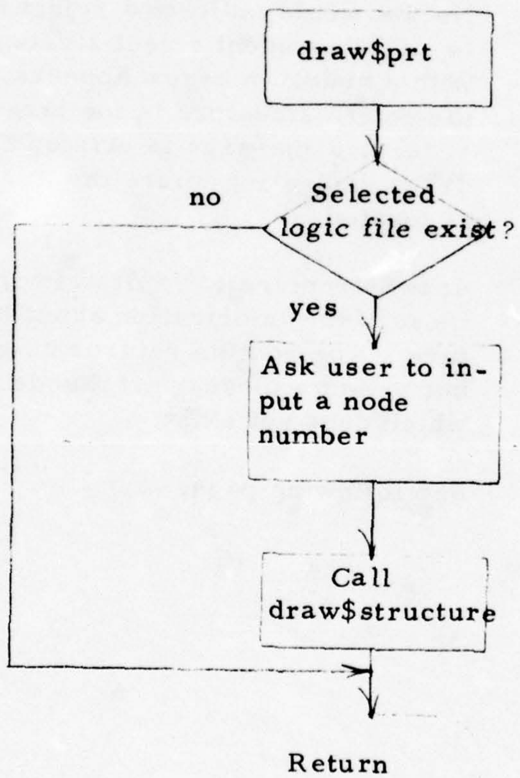
Calling Sequence: Type "draw$prt [(treename)][(nodename)] "

Input Parameters: Standard optional data set selection parameters

Input File Settings: A mooslogic file must be present in the process directory.

Program Description: draw$prt asks the user to select a logic node number and calls draw$*structure*, which displays the selected logic file structure below the selected node as a tree.

Flow Chart:



Internal Subroutine Name: draw$structure

Calling Sequence: call draw$structure (lptr, m)

Input Parameters:

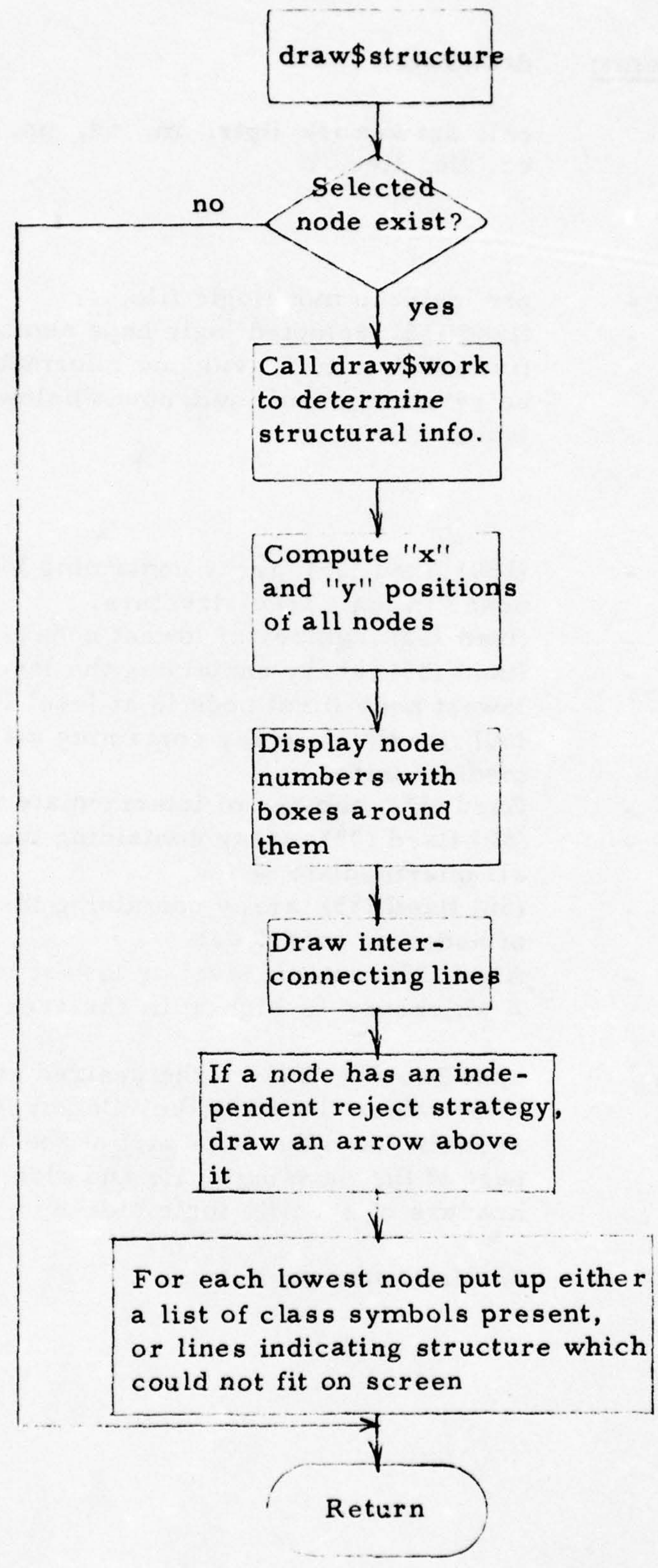
lptr	-	prt pointer to a mooslogic file.
m	-	fixed (35) logic node number.

Program Description:

draw$structure displays a given logic tree below the selected node. Each logic node is displayed as a rectangular box containing the node number. If there is no structure below a node, a list of class symbols present at that node appears to the right of the node. No list would indicate a reject node. If there is an independent reject strategy associated with a node, an arrow appears above it. If the entire structure is too large to fit on the screen, a message is printed and lines are drawn indicating where the missing structure is located.

draw$structure calls draw$work to determine most of the information about logic tree structure. The routine returns no error codes, but prints a message if a node is selected which does not exist.

Flow Chart: See following page.


draw$structure
Selected node exist?
no
yes
Call draw$work to determine structural info.
Compute "x" and "y" positions of all nodes
Display node numbers with boxes around them
Draw inter-connecting lines
If a node has an independent reject strategy, draw an arrow above it
For each lowest node put up either a list of class symbols present, or lines indicating structure which could not fit on screen
Return

Internal Subroutine Name: draw$work

Calling Sequence: call draw$work (lptr, m, nd, nn, lln, cm, cc, llc, llvc, l)

Input Parameters:

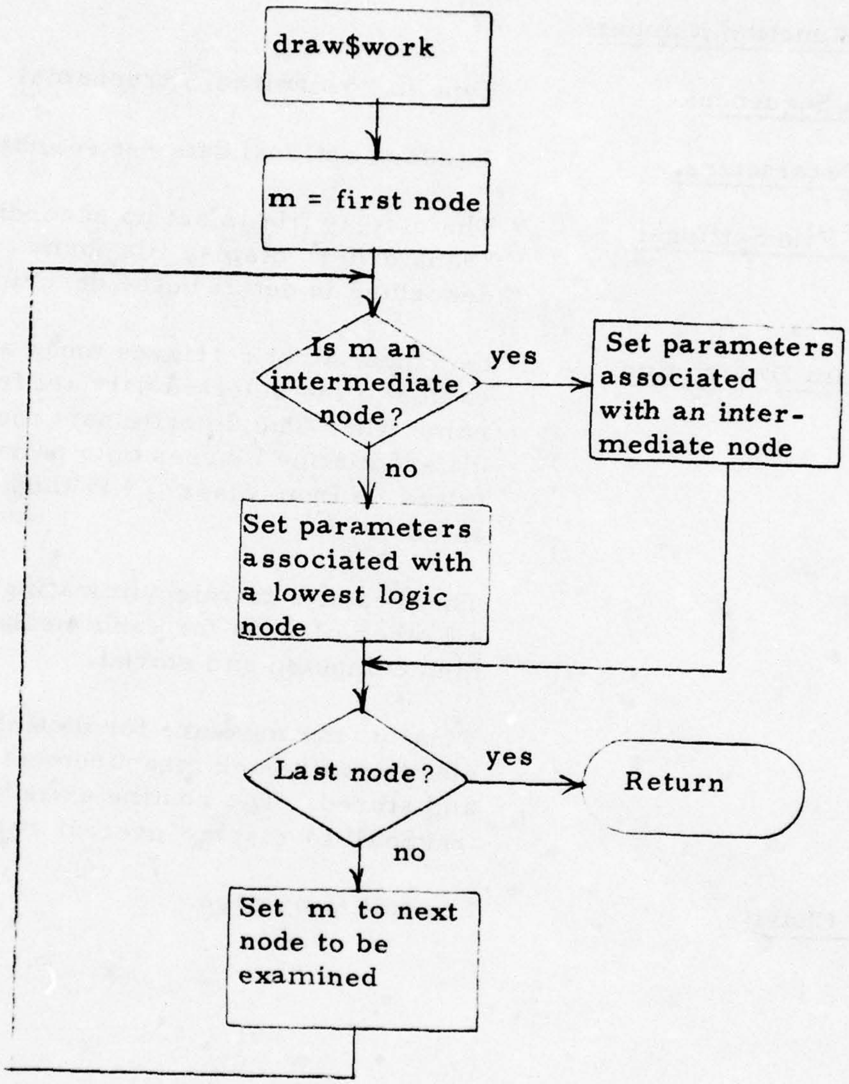
lptr	-	ptr point to mooslogic file.
m	-	fixed (35) selected logic node number.
l	-	fixed (35) cutoff level: no information will be returned about logic nodes below this level.

Output Parameters:

nd	-	(150) fixed (35) array containing lowest nodes in logic tree structure.
nn	-	fixed (35) number of lowest nodes.
lln	-	fixed (35) array containing the level of each lowest node (first node is at level 1).
cm	-	(60) fixed (35) array containing all intermediate nodes.
cc	-	fixed (35) number of intermediate nodes.
llc	-	(60) fixed (35) array containing the levels of all intermediate nodes.
llvc	-	(50) fixed (35) array containing the number of nodes on each level.
l	-	fixed (35) cut-off level or lowest level found - whichever is higher in the tree structure.

Program Description: draw$work retrieves the desired structural information (listed in the "Output Parameters" section) from the node part of the structure part of the mooslogic file and also from the headers of specific logic blocks.

Flow Chart: See following page.


draw$work
m = first node
Is m an intermediate node?
yes
Set parameters associated with an inter-mediate node
no
Set parameters associated with a lowest logic node
Last node?
yes
Return
no
Set m to next node to be examined

MOOS Function Name: dscrmeas

MOOS Function Number: 27

Calling Sequence: Type in "dscrmeas [(treename) [(nodename)] "

Input Parameters: Standard optional data set selection parameters.

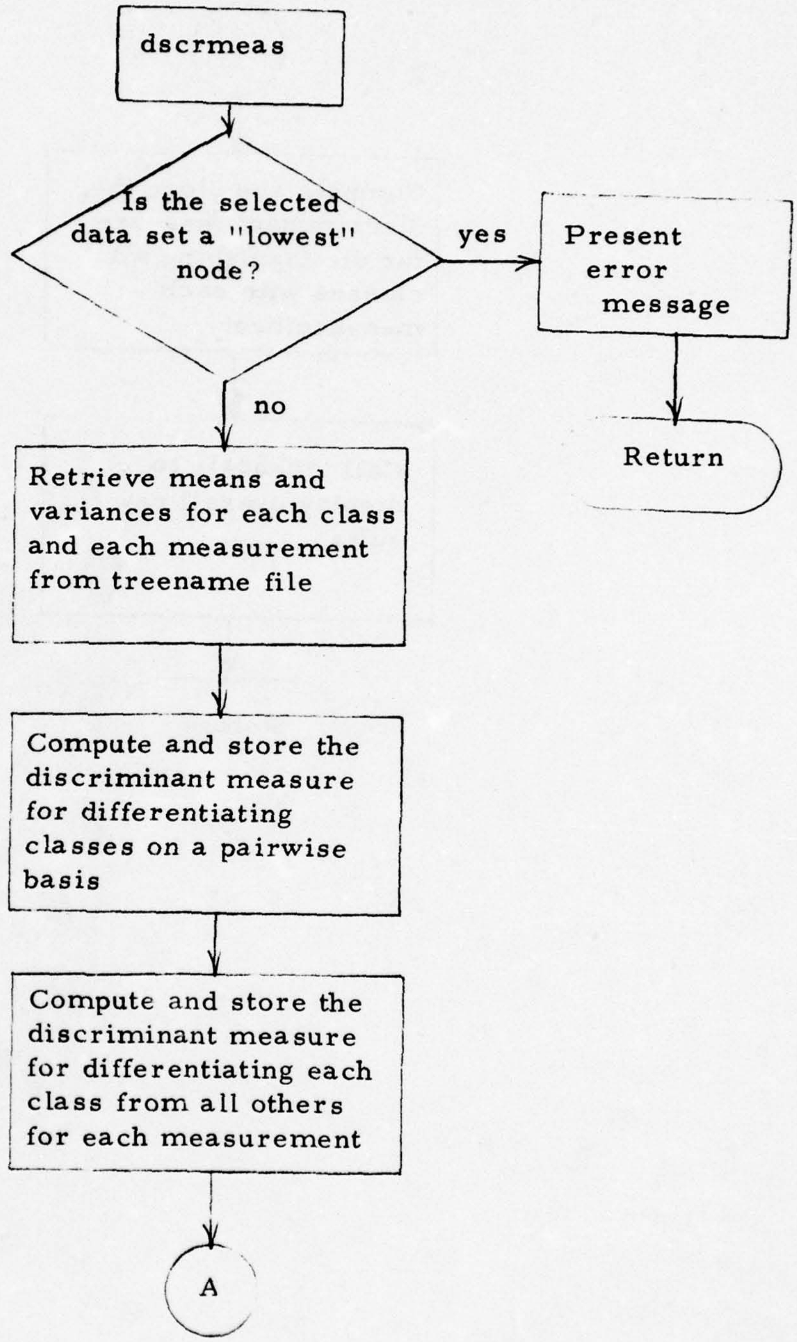
Output File Settings: The display file is set up according to the "rank order" display file format which is described in detail in the description of rnk.

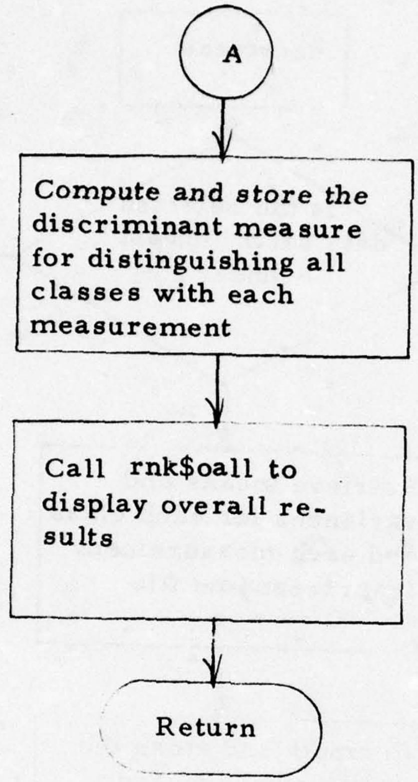
Program Description: dscrmeas first retrieves mean and variance values of the selected data set from the treename file. The discriminant measure for differentiating classes on a pairwise basis (class i from class j) is then calculated and stored.

The measure for discriminating class i from all other classes for each measurement is then computed and stored.

Finally, the measure for distinguishing all classes with each measurement is computed and stored. The routine exits by calling rnk$oall to display overall results.

Flow Chart: See following page.


dscrmeas
Is the selected data set a "lowest" node?
yes
Present error message
Return
no
Retrieve means and variances for each class and each measurement from treename file
Compute and store the discriminant measure for differentiating classes on a pairwise basis
Compute and store the discriminant measure for differentiating each class from all others for each measurement
A


A
Compute and store the discriminant measure for distinguishing all classes with each measurement
Call rnk$oall to display overall results
Return

MOOS Function Name: dsubstrc

MOOS Function Number: 21

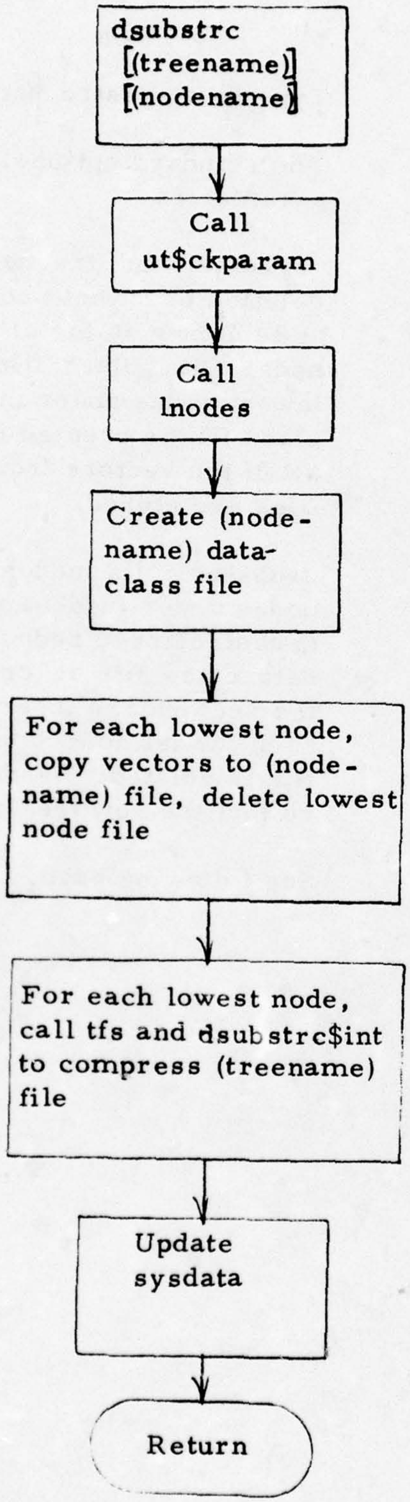
Calling Sequence: Type in "dsubstrc [(treename)] [)nodename)] "

Input Parameters: The standard optional data set selection parameters

Output File Settings: "sysdata" and (treename) file reflect the deletion of lowest nodes under (nodename) and the new status of (nodename) as a lowest node. Data class files are deleted for all lowest nodes under (nodename), and a data class file is created for (nodename) in which all of the vectors from the deleted data class files are stored.

Program Description: dsubstrc calls lnodes to determine the lowest nodes under (nodename), then calls tfs and dsubstrc$int to reduce the treename file. A data class file is created for (nodename), and the vectors are inserted into this file from each lowest node data class file before the file is deleted. Sysdata is then adjusted to reflect the new tree structure.

Flow Chart: See following page.


dsubstrc
[(treename)]
[(nodename)]
Call
ut$ckparam
Call
lnodes
Create (node-
name) data-
class file
For each lowest node,
copy vectors to (node-
name) file, delete lowest
node file
For each lowest node,
call tfs and dsubstrc$int
to compress (treename)
file
Update
sysdata
Return

Internal Subroutine Name: dsubstrc$int

Calling Sequence: call dsubstrc$int (tptr, index)

Input Parameters:

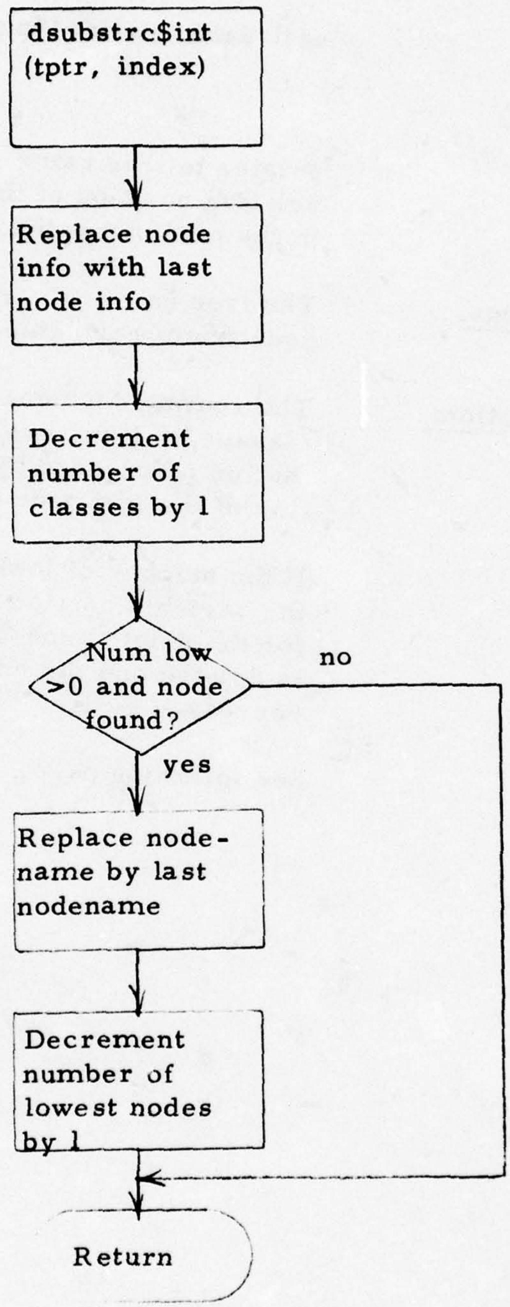
- tptr - pointer to tree name file.
- index - relative position of the node to be deleted in the tree name file

Output File Settings: The tree name file reflects the deletion of a node from tree name file.

Program Description: The routine decrements the number of classes by 1 and overwrites the node information referenced by index with the information of the last node in the file.

If the number of lowest nodes is not zero, the variable portion of the file is searched for the deleted node name. If it is found, it is deleted and the number of lowest nodes is decreased by 1.

Flow Chart: See following page.


dsubstrc$int (tptr, index)
Replace node info with last node info
Decrement number of classes by 1
Num low >0 and node found?
no
yes
Replace node-name by last nodename
Decrement number of lowest nodes by 1
Return

Programmer Aid Name: dump

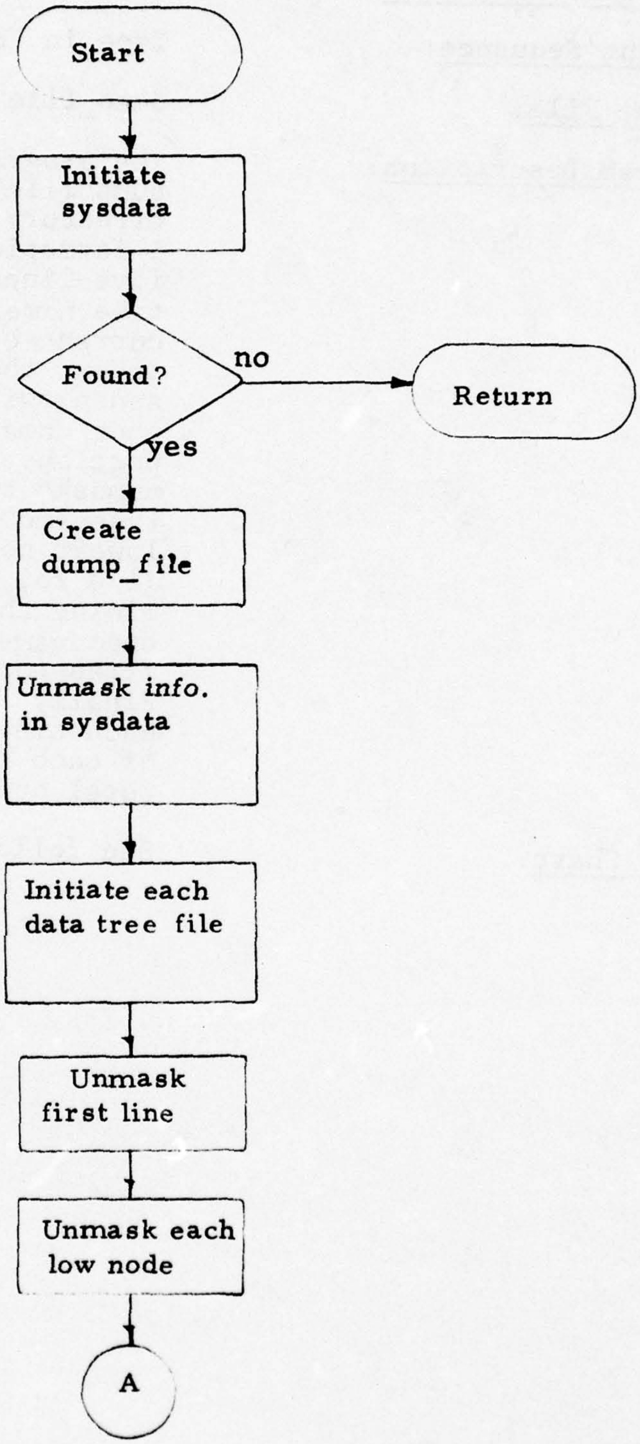
Calling Sequence: Type in "dump"

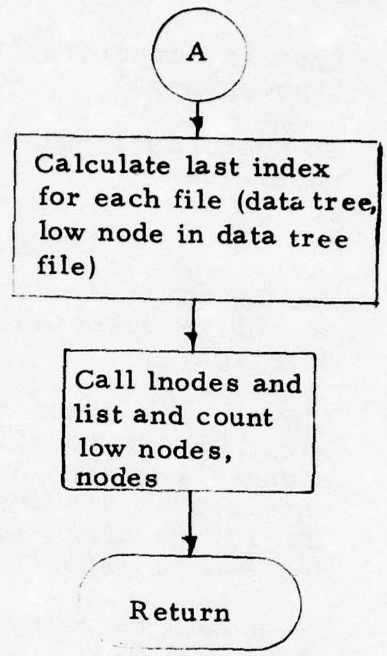
Output File: dump_file

Program Description: The program first creates a dump_file in the user's login directory, then inserts sysdata information, unmasking the first five lines to obtain the current tree name and tree index, the current class name, the last node index, the current option and sense switch settings. "dump" then unmasks the forest and school portions. Next, the routine unmasks the first line of each treename file, listing all the lowest nodes as registered below line 26, and in addition determining the last index for the tree name and those data class files for dumping purposes. Finally "lnodes" is called for each tree, and a count and listing of each tree's lowest nodes and total nodes is generated.

Flow Chart: See following page

dump


Start
Initiate sysdata
Found?
no
Return
yes
Create dump_file
Unmask info. in sysdata
Initiate each data tree file
Unmask first line
Unmask each low node
A


A
Calculate last index for each file (data tree, low node in data tree file)
Call lnodes and list and count low nodes, nodes
Return

MOOS Function Name: dvectors

MOOS Function Number: 40

Calling Sequence: Type in "dvectors [(treename)] [(nodename)]"

Input Parameters: Standard optional data set selection parameters

Output File Settings:

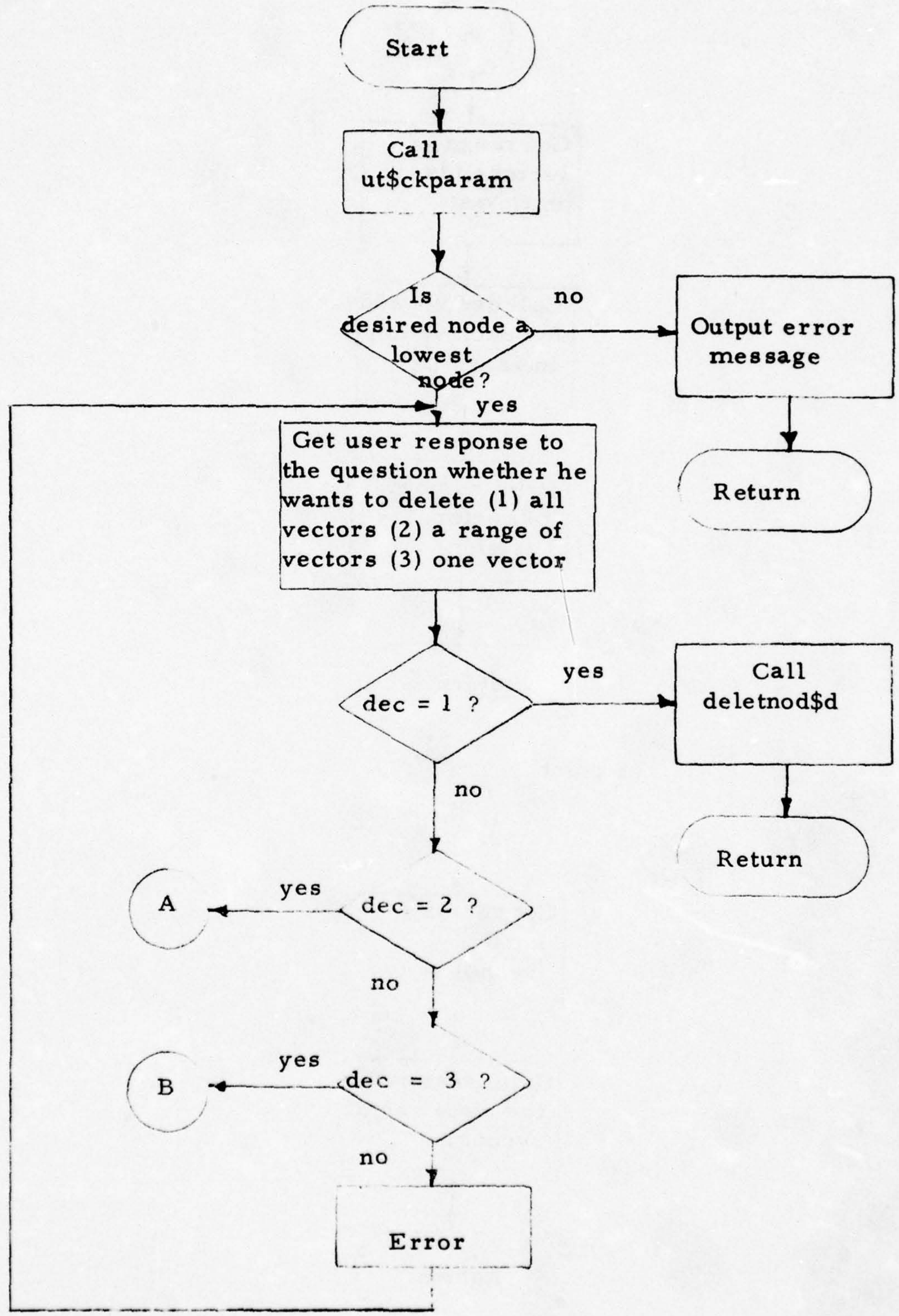
Dataclass File - (nodename): The associated vector(s) are deleted and D1 is decremented to reflect this deletion.

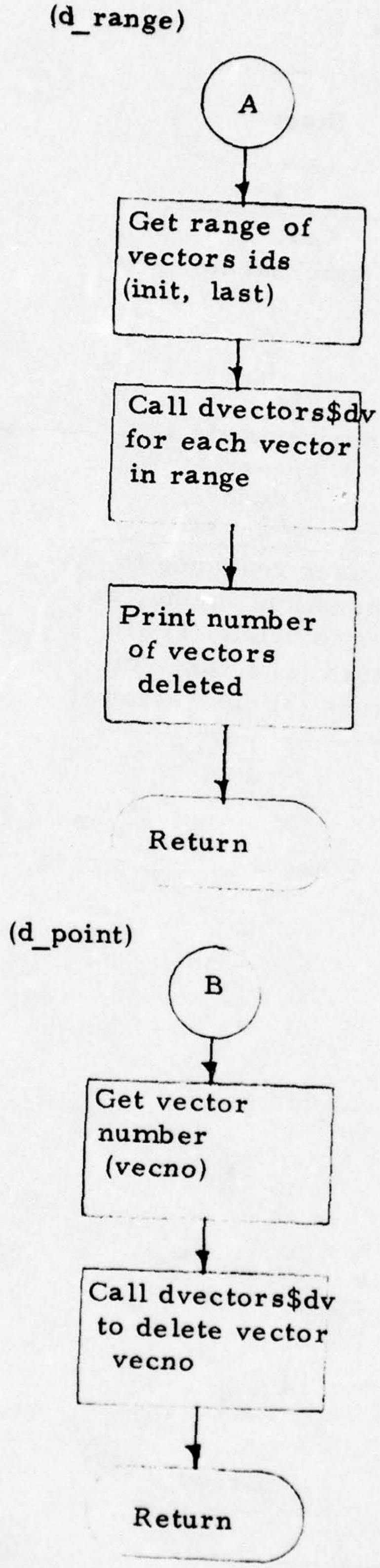
Treename File: All entries above the nodename in the datatree are modified to show the changes in means and covariance matrices as a result of the deletion. CM2 for each node involved is decremented appropriately.

Sysdata: S5 and/or F5 is modified for each node above nodename in the data tree to reflect the deletion.

Program Description: "dvectors" deletes vectors from a data class file. The program first asks the user if he wants to delete all vectors, a range of vectors, or 1 vector. (1) If all, internal subroutine "deletnod$d" is called to delete the node. (2) If a range of vectors, the program asks for the initial and last vector id number (inclusive) to be deleted. The program then calls internal subroutine "dvectors$dv" to delete each vector in the range. (3) If one vector is specified, the program asks for the vector id number and calls routine "dvectors$dv" to delete the vector. (In the latter two cases, if during the process of deletion the user has specified to delete the only remaining vector for the class, the user is informed as such and is asked whether he still wants to delete it. If yes, subroutine "deletnod$d" is called and the entire node is deleted. If no, the program returns and all vectors up to that point are deleted.)

Flow Chart: dvectors




(d_range)
A
Get range of vectors ids (init, last)
Call dvectors$dv for each vector in range
Print number of vectors deleted
Return
(d_point)
B
Get vector number (vecno)
Call dvectors$dv to delete vector vecno
Return

Internal Subroutine Name: dvectors$cvmean

Calling Sequence: call dvectors$cvmean (treeptr, ix, vdata)

Input Parameters:

treeptr — a ptr to the top of the associated treename file. This is the file where modification of means and covariance matrix will take place. [ptr]

ix — a relative index from "treeptr" to the CM1 entry of the associated node to be modified [fixed bin (35)]

vdata — an ndim dimensional array which contains the vector, which will cause the modifications. [float]

Output File Settings: The associated treename file is modified to show the result of removing the vectors.

Program Description: "dvectors$cvmean" modifies the mean and covariance matrix for a given node to reflect the deletion of a vector from a node.

Computation is done as follows:

Let μ_i = i^{th} component of the mean before modification

μ_i' = i^{th} component of the mean after modification

σ_{ij} = $(i,j)^{th}$ component of the covariance matrix before modification

σ_{ij}' = $(i,j)^{th}$ component of the covariance matrix after modification

N = number of vectors before modification

N' = number of vectors after modification

X_i = i^{th} component of the vector

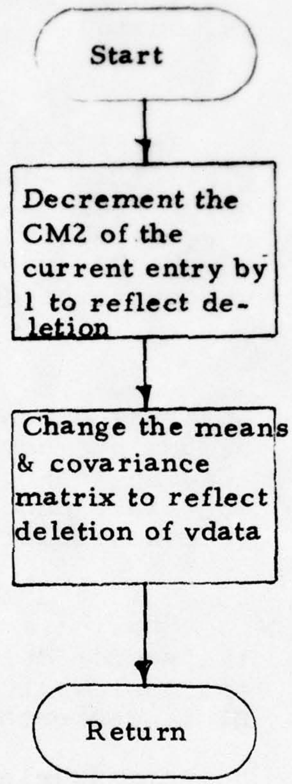
Then $N' = N-1$

$$\mu_i' = \frac{N\mu_i - X_i}{N-1}$$

$$\sigma_{ij}' = \left[\frac{N\,(\sigma_{ij} + \mu_i \mu_j) - X_i X_j}{N-1}\right] - \mu_i' \mu_j'$$

Flow Chart: See following page

dvectors$cvmean



Internal Subroutine Name: dvectors$delete

Calling Sequence: call dvectors$delete (classptr, ix, vdata)

Input Parameters:

classptr - a pointer to the appropriate data class file where deletion will occur [ptr]

ix - a relative index from "classptr" to the D3 entry of the associated vector to be deleted [fixed bin(35)]

Output Parameter:

vdata - an array dimensioned to the dimensionality of the data. Upon leaving the routine, "vdata" will contain each component of the vector that was deleted [float]

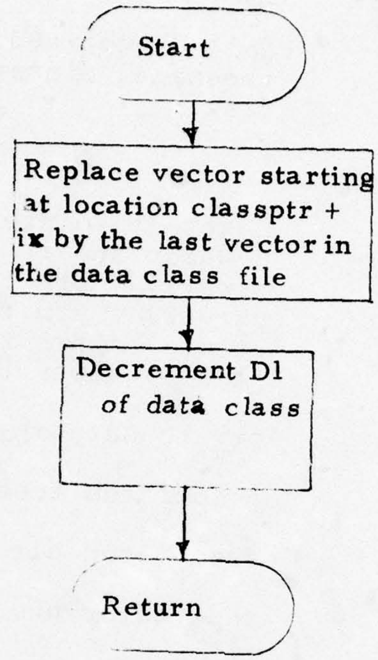
Output File Settings:

dataclass file - the vector deleted is replaced by the last vector in the data class file. D1 is decremented by 1.

Program Description: "dvectors$delete" deletes one vector from a data class file by replacing it by the last vector in the data file.

Flow Chart: See following page

dvectors$delete



Internal Subroutine Name: dvectors$dv

Calling Sequence: call dvectors$dv (ptrs, classptr, treename, classname, vecno, vdata, flag)

Input Parameters:

ptrs: a set of pointers dimensioned to 4 which are :
ptrs(1) = ptr to sysdata file
ptrs(2) = ptr to scratch file
ptrs(3) = ptr to display file
ptrs(4) = ptr to tree name file

classptr: ptr to data class file

treename: associated tree name

classname: associated class name

vecno: the vector number to be deleted

flag: a fixed binary (1) number, which if "1) implies communication with the user will take place during deletion if the last vector of a data class takes place. If "0" no communication occurs.

Output Parameters:

vdata: an array dimensioned to 50 which will contain the vector data of the vector being deleted.

flag: set to 1 if the desired vector was deleted. 0 if not.

Output File Settings: The vector is deleted from the data class file. D1 is also decremented by 1.

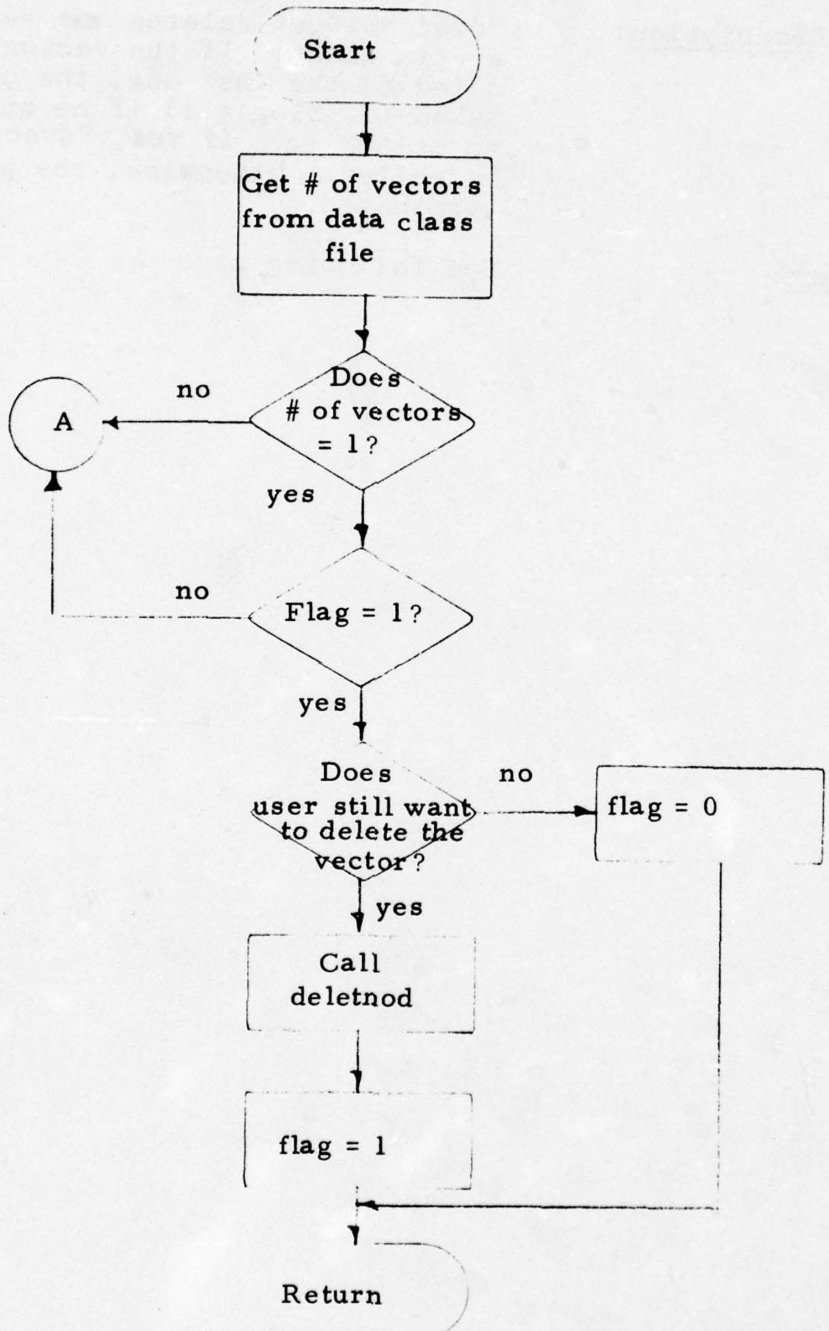
treename file: All entries including and above class name in the data tree are modified to show the change in means and co-variance matrix as a result of the deletion. CM2 for each node involved is decremented by 1.

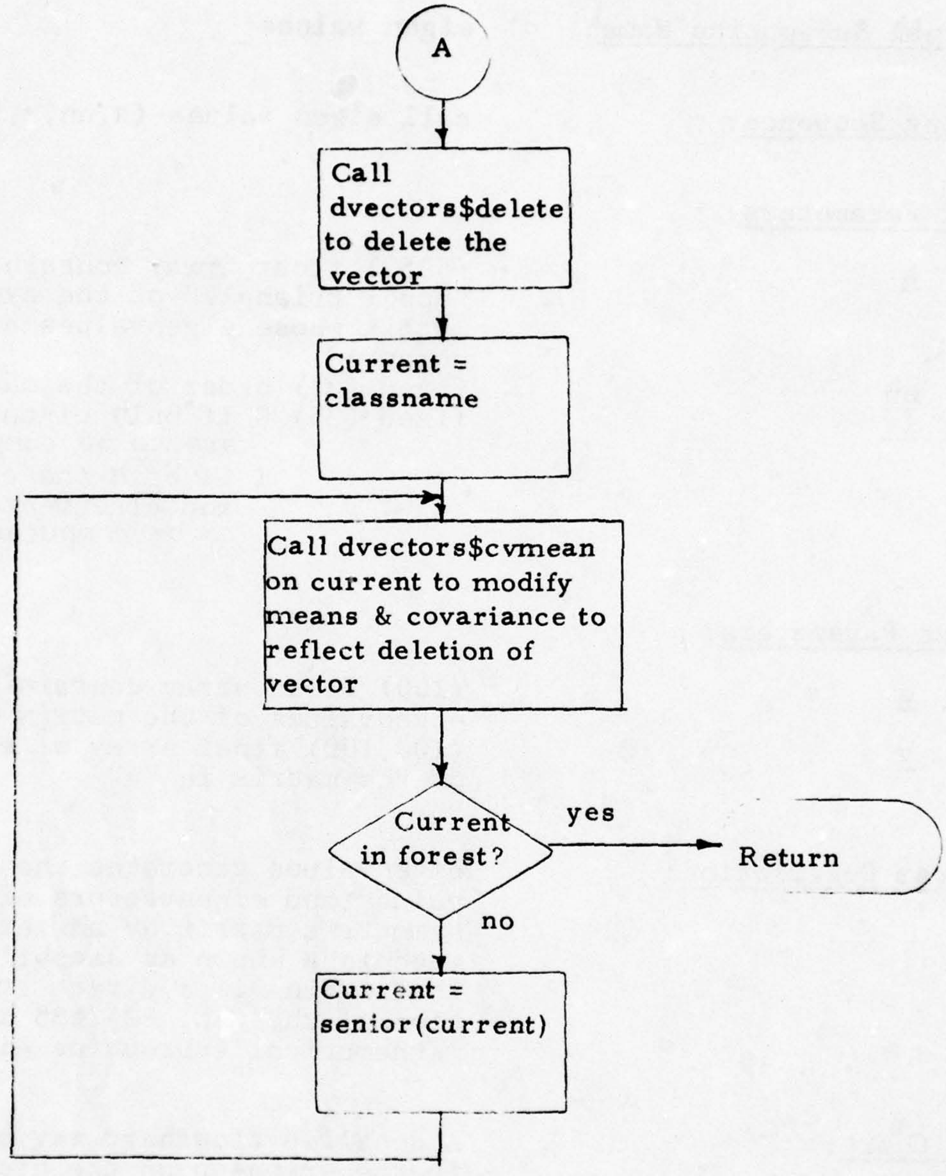
sysdata — S5 or F5 is decremented by 1 for each node above class name in the data tree.

Program Description: — "dvectors$dv" deletes one vector from a data class. If the vector to be deleted is the last one, the user is asked (if flag = 1) if he still wants to delete it. If yes, "dvectors$delete" is called. Otherwise, the program is terminated.

Flow Chart: — See following page

dvectors$dv


Start
Get # of vectors from data class file
Does # of vectors = 1?
no
A
yes
no
Flag = 1?
yes
Does user still want to delete the vector?
no
flag = 0
yes
Call deletnod
flag = 1
Return


A
Call dvectors$delete to delete the vector
Current = classname
Call dvectors$cvmean on current to modify means & covariance to reflect deletion of vector
Current in forest?
yes
Return
no
Current = senior(current)

Internal Subroutine Name: eigen_values

Calling Sequence: call eigen_values (a,nn,u,iv,e)

Input Parameters:

a — (5050) float array containing the "upper triangle" of the symmetric matrix whose eigenvalues are to be computed.

nn — fixed (35) order of the matrix in "a"

iv — fixed (35) 0 if only eigenvalues are to be computed.
1 if both the eigenvalues and eigenvectors are to be computed.

Output Parameters:

e — (100) float array containing the eigenvalues of the matrix in "a"

v — (100,100) float array of eigenvectors of the matrix in "a"

Program Description: eigenvalues generates the eigenvalues and eigenvectors of a symmetric matrix by an interative technique known as Jacobi's method. The routine is a direct translation of the G.E. 625/635 series mathematical subroutine eigenj.

Flow Chart: A detailed flowchart may be found in the write-up on the previously mentioned subroutine eigenj.

Internal Subroutine Name: eigenp

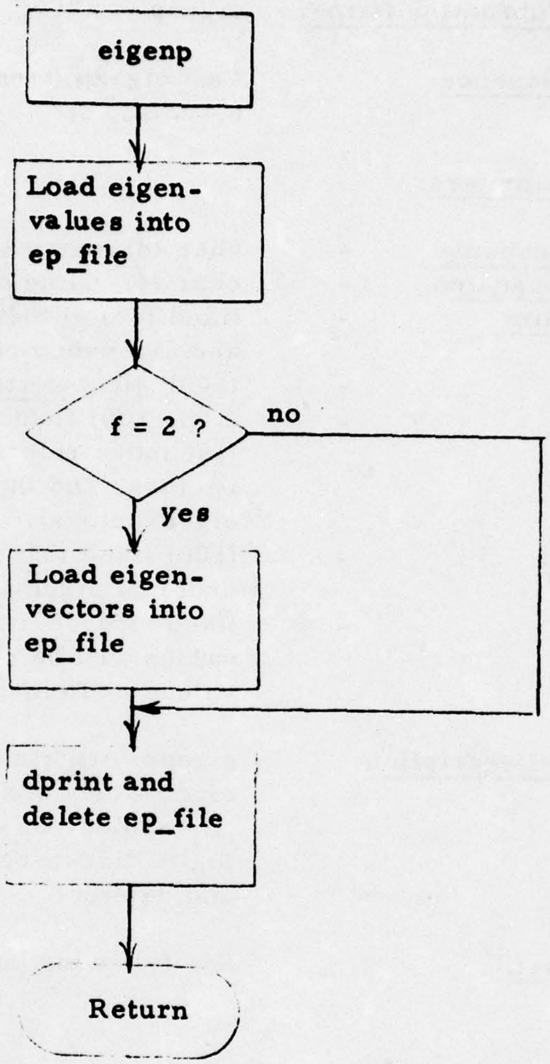
Calling Sequence: Call eigenp (treename, nodename, ndim, e, v, so, f)

Input Parameters:

treename	-	char (8) name of data tree.
nodename	-	char (4) name of data node.
ndim	-	fixed (35) dimensionality of eigenvalues and eigenvectors.
e	-	(100) float sorted eigenvalues.
v	-	(100, 100) float unsorted eigenvectors (1st index refers to elements of eigenvectors, 2nd index refers to specific eigenvectors).
so	-	(100) fixed (35) array containing sort order of eigenvalues.
f	-	fixed (35) if f is set to 1, only eigenvalues will be printed; if 2, both eigenvalues and eigenvectors are printed.

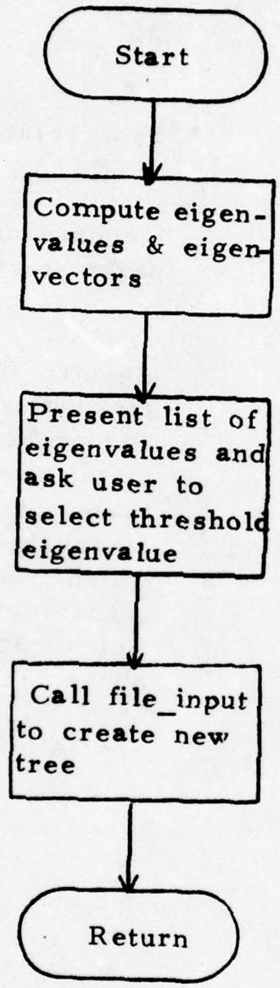
Program Description: eigenp formats the input eigenvectors and eigenvalues and loads them in an output file named "ep_file" located in the users login directory. This file is then "dprinted" and deleted.

Flow Chart: See following page.


eigenp
Load eigen-
values into
ep_file
f = 2 ?
no
yes
Load eigen-
vectors into
ep_file
dprint and
delete ep_file
Return

MOOS Function Name:	eigentrn
MOOS Function Number:	128
Calling Sequence:	Type "eigentrn (treename) "
Input Parameters:	standard optional data set selection parameters
Program Description:	eigentrn first computes the eigenvalues of the selected data set and presents a list of eigenvalues to to the user. The user is then asked to select a cutoff or threshold eigenvalue. A new tree is created which consists of the original tree transformed by the eigenvectors which correspond to the eigenvalues above the threshold.
Flow Chart:	See following page.

eigentrn



Start
Compute eigen-values & eigen-vectors
Present list of eigenvalues and ask user to select threshold eigenvalue
Call file_input to create new tree
Return

MOOS Function Name: eigv$sa2

MOOS Function Number: 201

Calling Sequence: Type in "eigv$sa2 [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: eigv$sa2 calls eigv$eigen to project the selected data set on any two eigenvectors of the data set.

MOOS Function Name: eigv$sal

MOOS Function Number: 215

Calling Sequence: Type in "eigv$sal [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: eigv$sal calls eigv$eigenl to project the selected data set on any eigenvector of the data set.

MOOS Function Name: eigv$ld2

MOOS Function Number: 70

Calling Sequence: Type in "eigv$ld2 [(treename)] (nodename) "

Input Parameters: Standard optional data set selection parameters

Program Description: eigv$ld2 calls eigv$eigen to project the selected data set on any two eigenvectors of the data set. Logic may be created from the resulting display.

MOOS Function Name: eigv$ldl

MOOS Function Number: 85

Calling Sequence: Type in "eigv$ldl (treename) (nodename) "

Input Parameters: Standard optional data set selection parameters

Program Description: eigv$ldl calls eigv$eigenl to project the selected data set on any eigenvector of the data set. Logic may be created from the resulting display.

Internal Subroutine Name: eigv$eigen
eigv$eigenl

Calling Sequence: call eigv$eigen (ptrf, x, trnam, nod)
call eigv$eigenl (ptrf, x, trnam, nod)

Input Parameters:

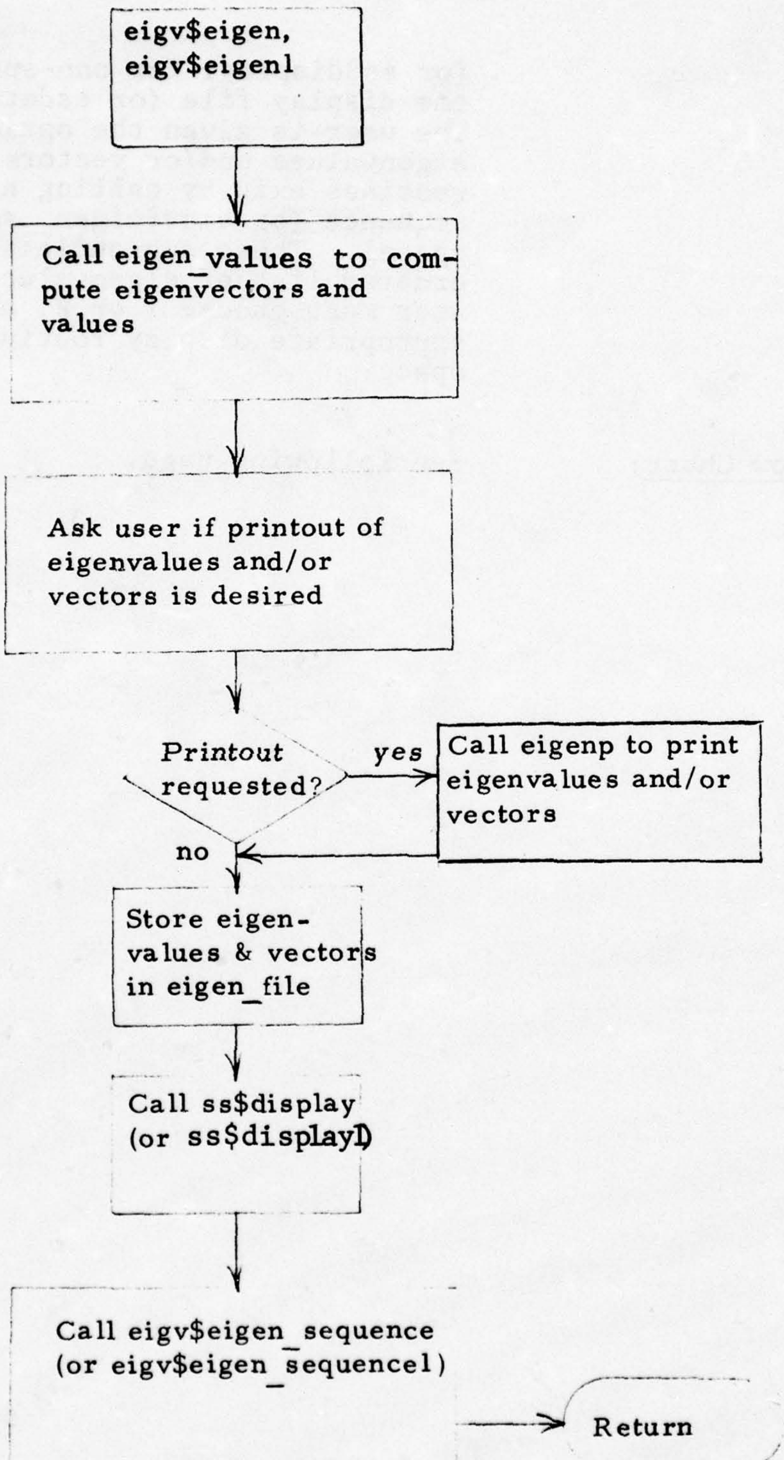
ptrf	-	(5) ptr	ptrf(1) - sysdata ptrf(2) - scratch ptrf(3) - display ptrf(4) - treename ptrf(5) - mooslogic
x	-	fixed (35) x should be set to 1 for logic design, 0 for structure analysis.	
trnam	-	char (8) tree name of selected data set.	
nod	-	char (4) node name of selected data set.	

Output File Settings: Entry eigen sets up display for a two space plot by calling ss$display, Entry eigenl sets up csdata for a one-space plot by calling ss$displayl. Both routines store eigenvalues and vectors in a process directory file named eigen_file.

Program Description: eigen and eigenl call eigenvectors and eigenvalues for the selected data set. These values are then stored in the users process directory in a file called eigen_file. The program then calls ss$display

(or ss$displayl for one-space) to initialize the display file (or csdata for one-space). The user is given the option of having eigenvalues and/or vectors printed. The routines exit by calling eigv$eigen_ sequence (or eigv$eigen_ sequencel for one-space). These subroutines present an ordered list of eigenvalues from which the user must choose 1 or 2, and call the appropriate display routine for one or two-space.

Flow Chart: See following page.

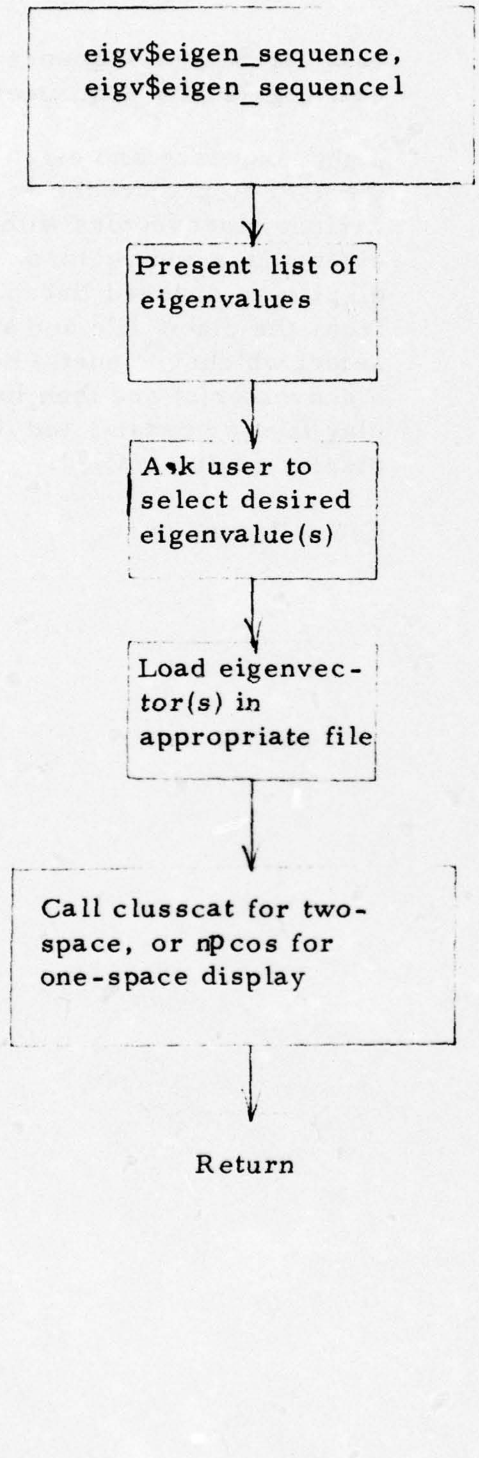

eigv$eigen,
eigv$eigenl
Call eigen_values to compute eigenvectors and values
Ask user if printout of eigenvalues and/or vectors is desired
Printout requested?
yes
Call eigenp to print eigenvalues and/or vectors
no
Store eigenvalues & vectors in eigen_file
Call ss$display (or ss$displayl)
Call eigv$eigen_sequence (or eigv$eigen_sequencel)
Return

Internal Subroutine Name: eigv$eigen_sequence
eigv$eigen_sequencel

Calling Sequence: call eigv$eigen_sequence
call eigv$eigen_sequencel

Program Description: eigen_sequence and eigen_sequencel allow the user to project the selected data set on various eigenvectors without recalculating eigenvalues and vectors. The programs display an ordered list of eigenvalues taken from the eigen_file and ask the user to select whichever one(s) he wants. The eigenvector(s) are then loaded into the display file (or csdata) and the appropriate display routine called.

Flow Chart: See following page.


eigv$eigen_sequence,
eigv$eigen_sequence1
Present list of eigenvalues
Ask user to select desired eigenvalue(s)
Load eigenvector(s) in appropriate file
Call clusscat for two-space, or npcos for one-space display
Return

Utility Function Name: elimclas

Calling Sequence: call elimclas (option [symbol 1] [symbol 2]...[symbol N])

Input Parameter:

option — indicates what initial display value all present classes are to have,either all classes are displayed or no class are displayed. The parameter is either "on" or "off".

symbol 1-symbol N — a list of class symbols indicating which classes are to be displayed or not displayed.

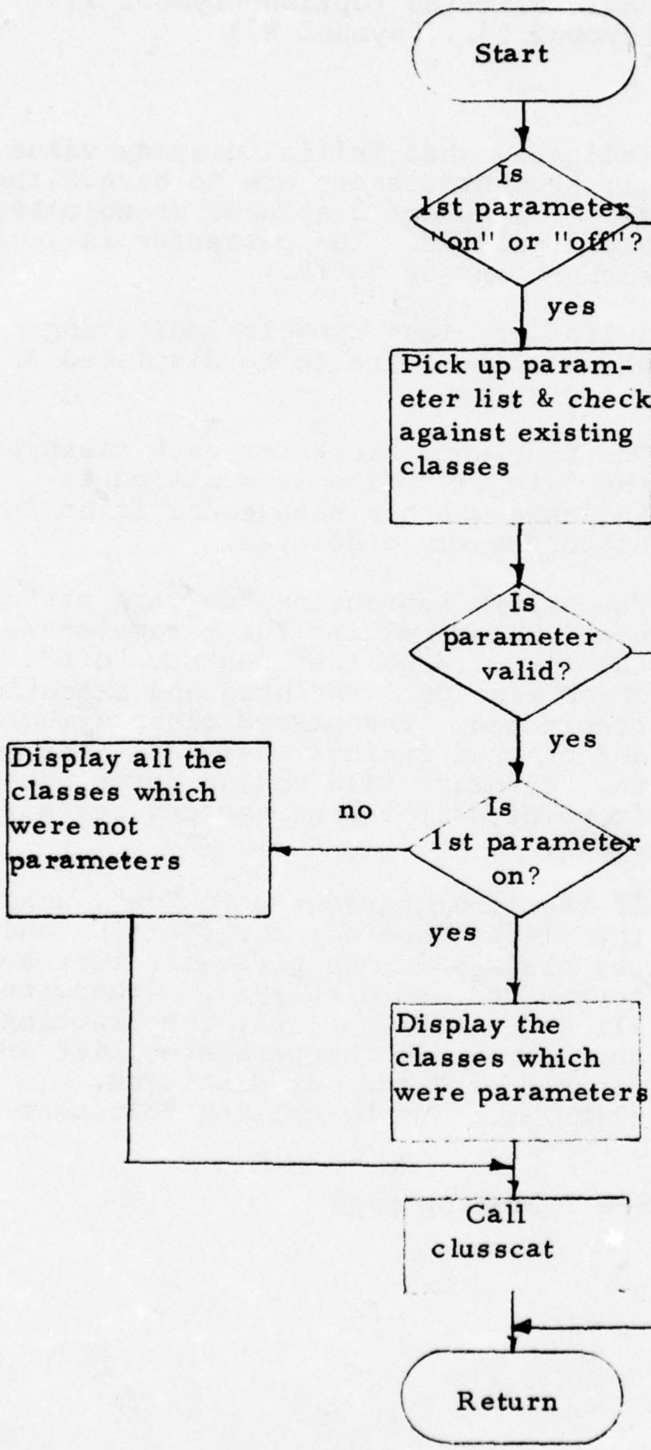
Output File Settings: The four-word block for each class of the "display" file is modified to indicate whether each class is or is not currently displayed.

Program Description: The system subroutine "cu_$arg_ptr" is used in determining the parameters. If the first is neither "on"nor "off", an error message is printed and execution terminated. The passed class symbols are checked against the class name in the "display" file and an error message is printed if the parameters are not valid.

If the first parameter is "on", none of the classes are set for plotting and the classes in the parameter list are "turned on" and displayed. Otherwise, all the classes are set for plotting and the classes in the parameter list are "turned off" and not displayed. elimclas exits by calling "clusscat."

Flow Chart: See following page

elimclas


Start
Is
1st parameter
"on" or "off"?
no
yes
Pick up parameter list & check against existing classes
Is
parameter
valid?
no
Print error message
yes
Display all the classes which were not parameters
no
Is
1st parameter
on?
yes
Display the classes which were parameters
Call clusscat
Return

Internal Subroutine Name: ellipse

Calling Sequence: call ellipse (logicptr, nodenum, index, ndim)

Input Parameters:

- logicptr — pointer to the MOOS logic file (ptr)
- nodenum — current logic node number (fixed (35))
- index — index to logic for this node (fixed (35))
- ndim — data dimensionality (fixed (35))

Program Description: ellipse generates FORTRAN code for a logic node using closed decision logic with the hyper-ellipsoid option

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Programmer Aid Name: fastdump

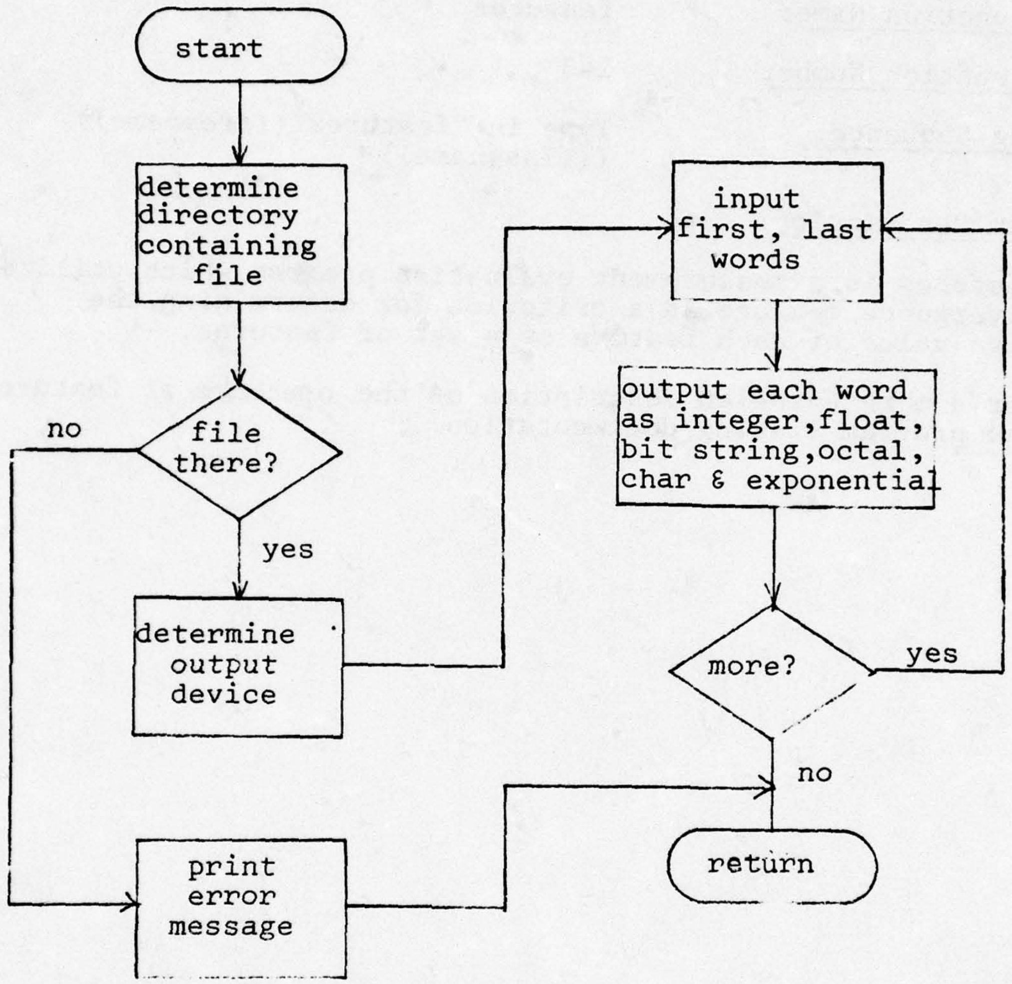
Programmer Aid Call: Type in "fastdump (file)"

Input Parameter: file or segment to be dumped

Program Description: fastdump asks the user where the file is located (working, login or process directory). When the file is found, the user is asked to specify either the line printer or the terminal as the device on which to print the output. Then the user inputs the first and last words of the section to be output. The value of each word is represented as an integer, a floating point number, a bit string, an octal number, a character string (with blanks representing unprintable characters), and an exponential number. The user may output more lines or may exit from the program, depending upon his answer to the question as to whether he wants to dump more.

Flow Chart: See following page.

fastdump


start
determine directory containing file
file there?
no
yes
determine output device
print error message
input first, last words
output each word as integer,float, bit string,octal, char & exponential
more?
yes
no
return

MOOS Function Name: features

MOOS Function Number: 143

Calling Sequence: Type in "features ((treename)) ((classname))"

Program Description:

features is a measurement evaluation program which utilizes the divergence measure as a criterion for determining the relative value of each feature of a set of features.

For a more detailed description of the operation of features, see the program listing documentation.

Utility Function Name: features_abs

Calling Sequence: Type in "features_abs"

Program Description:

features_abs creates a segment "f_abs_x.absin" (where x is a four-digit random number) in the user's home directory. In this segment features_abs places the various commands and answers to queries necessary to run program features. The dialogue of features_abs is designed to mimic the dialogue of program features. The output produced is identical to that which would be produced on-line by program features.

For a more detailed description of the operation of features_abs, see the program listing documentation.

Internal Subroutine Name: fileinput

Calling Sequence: call fileinput (treename)

Input Parameter:

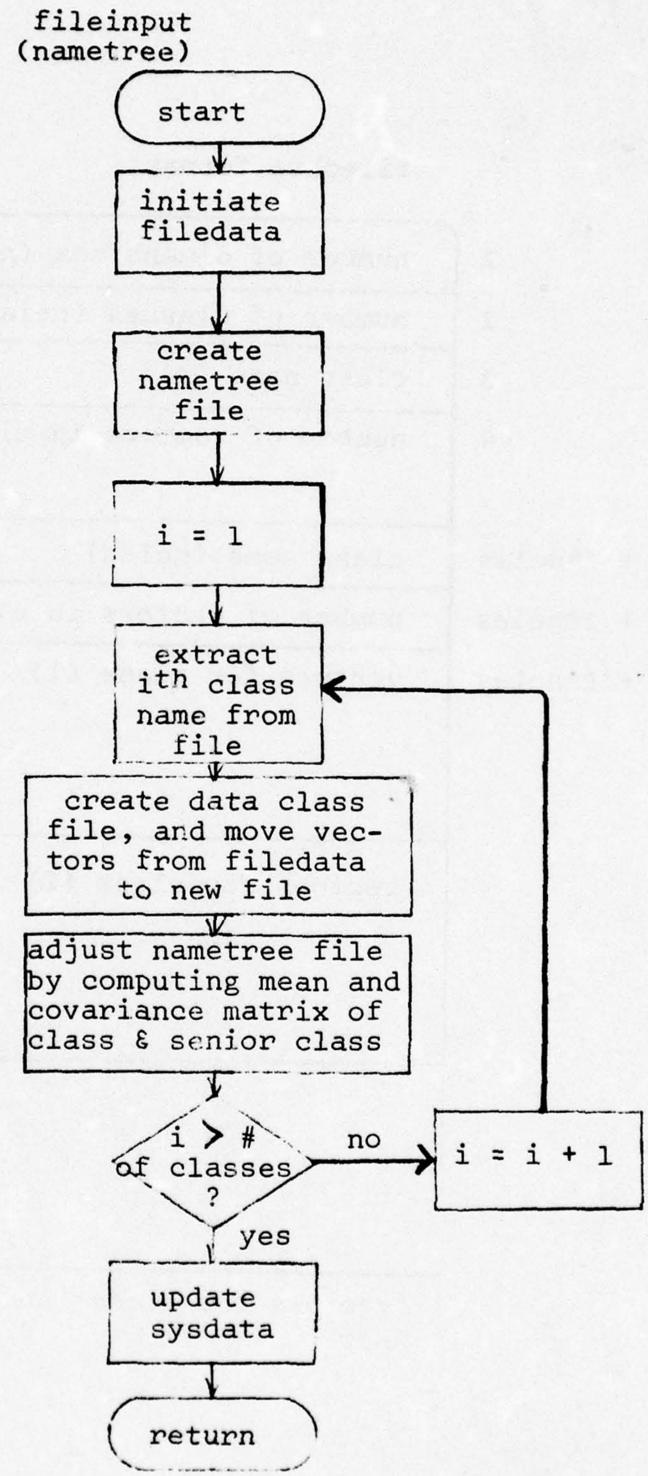
treename - an eight-character treename.

Input File Setting: The calling routine must create the file "filedata" in the process directory prior to the call.

Output File Settings: "sysdata" reflects the addition of a new tree in the system, a (treename) file is created, and appropriate values are inserted. Data-class files are created for each node and the appropriate vectors are stored in each.

Program Description: fileinput creates the tree (treename) from the calling program supplied file "filedata" in the process directory. The means and covariances are calculated as the vectors are stored in the dataclass files. The routine exits after displaying "no tree input" if "filedata" is not found.

Flow Chart: See following page.


fileinput
(nametree)
start
initiate filedata
create nametree file
i = 1
extract ith class name from file
create data class file, and move vectors from filedata to new file
adjust nametree file by computing mean and covariance matrix of class & senior class
i > # of classes ?
no
i = i + 1
yes
update sysdata
return

filedata format

1	number of dimensions (ndim)
2	number of classes (nclas)
3	class name (1)
4	number of vectors in class (1)
. . .	
1 + 2*nclas	class name (nclas)
2 + 2*nclas	number of vectors in class (nclas)
3 + 2*nclas	vectors for class (1) . . .
	vectors for class (2) . . .
	. . .
	vectors for class (nclas) . . .

MOOS Function Name: fisher

MOOS Function Number: 65

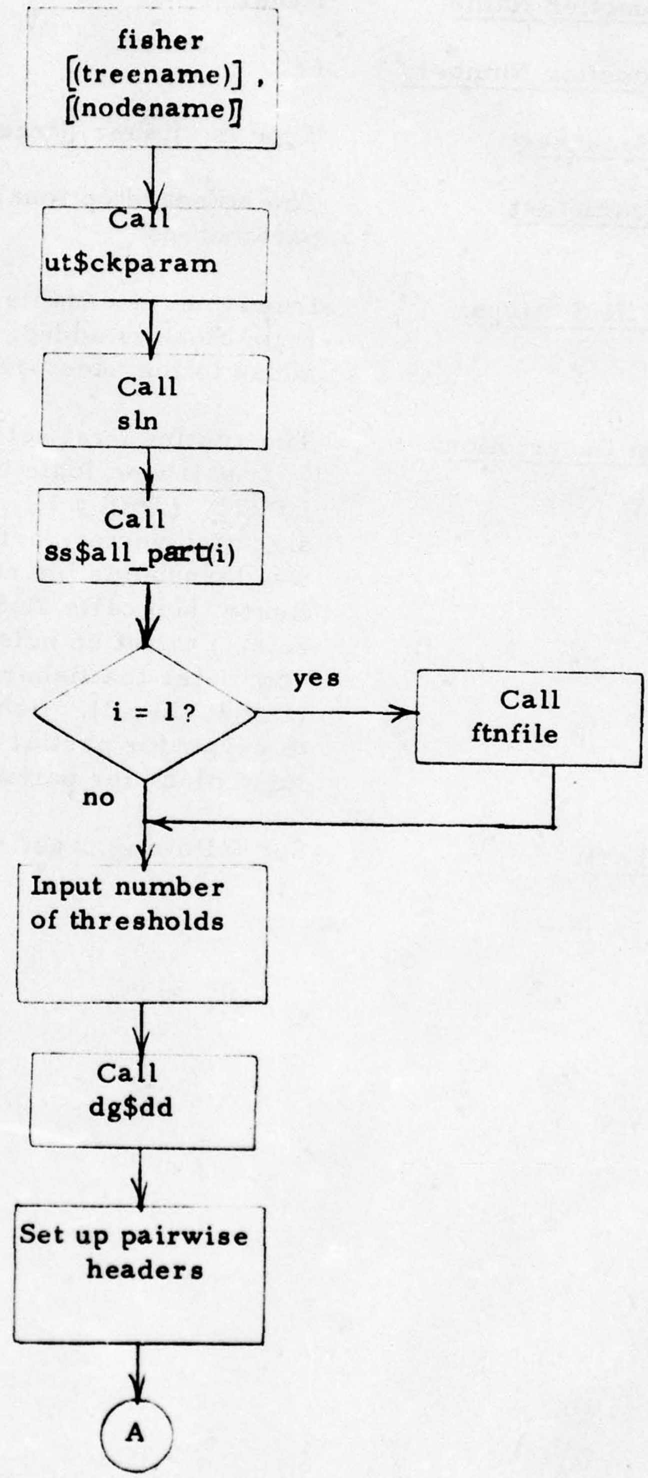
Calling Sequence: Type in "fisher [(treename)][(nodename)]"

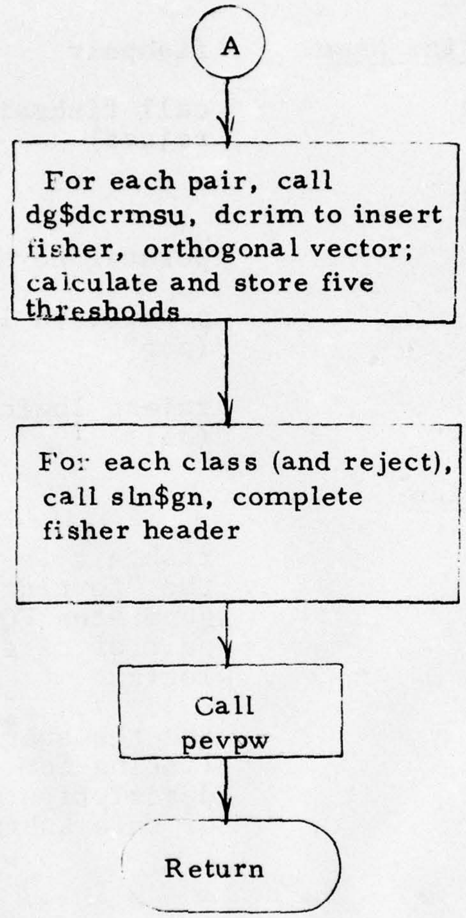
Input Parameters: The standard optional data set selection parameters

Output File Settings: treename, nodename mooslogic file: a fisher logic block is added, and ncls + 2 nodes are added to the node part of the structure part.

Program Description: The routine first sets up the [ncls x (ncls -1)] ÷ 2 pairwise logic block headers (LPR3 - LPR10, LPR6 = 1). It then inserts the fisher direction vector, orthogonal vector and the five thresholds for each pairwise block. fisher then calls sln$gn for each class (and reject) to set up ncls + 1 lowest nodes, and completes the fisher logic block header (LPR1, LPR2). fisher returns after a call to pevpw for partial evaluation. See the logic block for pairwise in Section 3.1

Flow Chart: See following page.


fisher
[(treename)] ,
[(nodename)]
Call
ut$ckparam
Call
sln
Call
ss$all_part(i)
i = 1?
yes
no
Call
ftnfile
Input number
of thresholds
Call
dg$dd
Set up pairwise
headers
A


A
For each pair, call dg$dcrmsu, dcrim to insert fisher, orthogonal vector; calculate and store five thresholds
For each class (and reject), call sln$gn, complete fisher header
Call pevpw
Return

Internal Subroutine Name: fishpair

Calling Sequence: call fishpair (logicptr, pairptr, reject)

Input Parameters:

logicptr — pointer to MOOS logic file (ptr)

pairptr — pointer to logic for this pair (ptr)

reject — reject logic node number (fixed (35))

Program Description:

fishpair is the subroutine in the "fortlogc" option which generates FORTRAN code for a pair of classes that uses fisher logic

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

MOOS Function Name: forteval

MOOS Function Number: 95

Calling Sequence:

From command level: Type in "forteval ((treename)) ((classname))"

From program level: call forteval (treename, classname)

treename: char (8) treename of the data set to be evaluated

classname: char (4) classname of the data set to be evaluated

Sense switch 8 should be set to indicate a call from program level, and CSS7 of sysdata should contain the name of the subroutine to be evaluated.

Output File Settings: The display file is set up in confusion matrix format

Program Description:

forteval tests a selected data set against a subroutine generated by fortlogc, or any subroutine with the same parameter list as a subroutine generated by fortlogc. Each vector in the selected data set is passed to the FORTRAN subroutine and the classification results are stored in the display file. The routine ends by calling conmatsm to display the confusion matrix.

For a more detailed description of the operation of forteval, see the program listing documentation.

Utility Function Name: fortlogc

Calling Sequence: Type in "fortlogc ((treename)) ((classname))"

Input Parameters: Standard optional data set selection parameters

Output File Settings: A FORTRAN subroutine is created in the user's login directory

Program Description:

fortlogc generates a FORTRAN subroutine which can classify data vectors according to the logic strategy of a specific MOOS logic tree. fortlogc calls logic_program to generate the source code, and then calls the compiler if the user requested a listing of the program or an evaluation. If the user requested an evaluation, forteval is called to perform the evaluation.

For a more detailed description of the operation of fortlogc, see the program listing documentation.

MOOS Function Name: fshp$sa2

MOOS Function Number: 203

Calling Sequence: Type in "fshp$sa2 [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: fshp$sa2 calls fshp$fish_pair which projects the selected data set on two fisher directions. The fisher directions are calculated for two user selected pairs of lowest nodes.

MOOS Function Name: fshp$ld2

MOOS Function Number: 71

Calling Sequence: Type in "fshp$ld2 [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Program Description: fshp$ld2 calls fshp$fish_pair which projects the selected data set on two fisher directions. The fisher directions are calculated for two user selected pairs of lowest nodes. Logic may be created from the resulting display.

Internal Subroutine Name: fshp$fish_pair

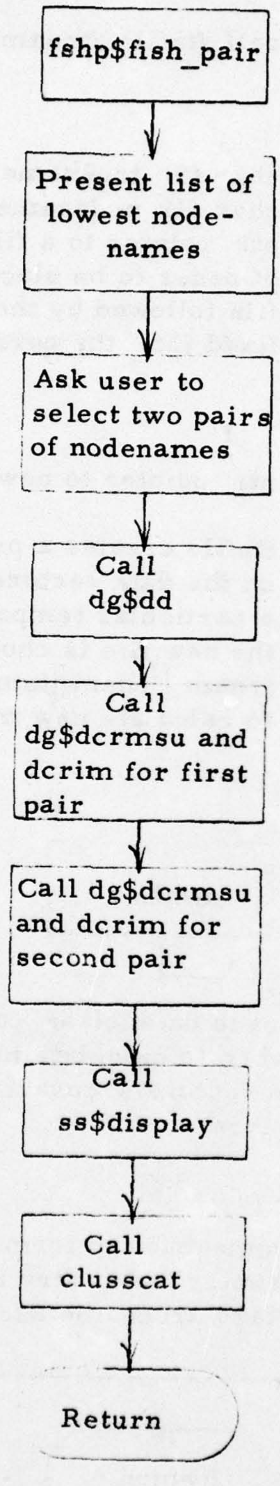
Calling Sequence: call fshp$fish_pair (ptrf, x)

Input Parameters:

ptrf - (5) ptr
- ptrf(1) - sysdata
- ptrf(2) - scratch
- ptrf(3) - display
- ptrf(4) - treename
- ptrf(5) - mooslogic

x - fixed (35) x must be set to 1 for logic design, 0 for structure analysis.

Output File Settings:	fshp$fish_pair sets up display for a two-space plot by calling ss$display.
Program Description:	fish_pair presents a list of lowest node names from the selected data set and asks the user to choose two pairs of node names. Fisher directions for each of these pairs are then calculated by separate calls to dg$dcrmsu and dcrim. The routine exits by calling the appropriate display routines.
Flot Chart:	See following page.


fshp$fish_pair
Present list of lowest node-names
Ask user to select two pairs of nodenames
Call dg$dd
Call dg$dcrmsu and dcrim for first pair
Call dg$dcrmsu and dcrim for second pair
Call ss$display
Call clusscat
Return

Internal Subroutine Name: ftnfile

Calling Sequence: call *ftnfile (trnam, cnam, nptr, temp, tptr)*

Input Parameters:

trnam	-	char (8) treename of data set.
cnam	-	char (4) nodename of data set
nptr	-	ptr pointer to a file containing the number of nodes to be placed in the new "treename" file followed by the names of these nodes.
temp	-	fixed (35) the current logic node number.

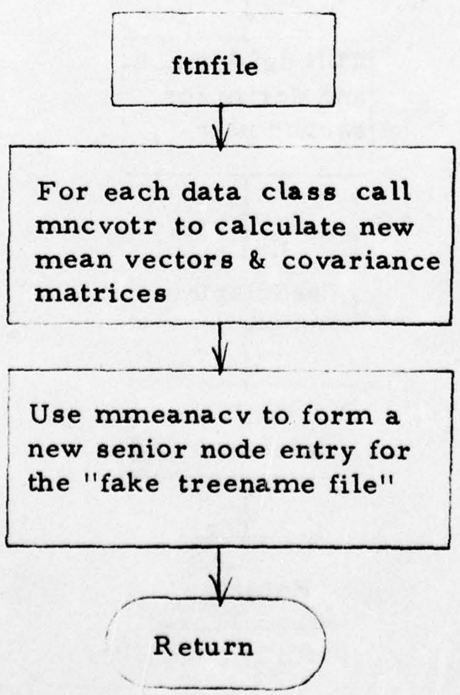
Output Parameters:

tptr	-	ptr pointer to new "tree

Program Description:

ftnfile creates a pseudo treename file based on the data vectors which are associated with a particular temporary symbol. The name of the new file is chosen as follows: trnam ||cnam||temp. uses mncvotr to calculate new means and covariances.

Flow Chart:



Internal Subroutine Name: getclass

Calling Sequence: call getclass (treename, nodename, index, array)

Input Parameters:

treename	associated tree name
nodename	associated class name

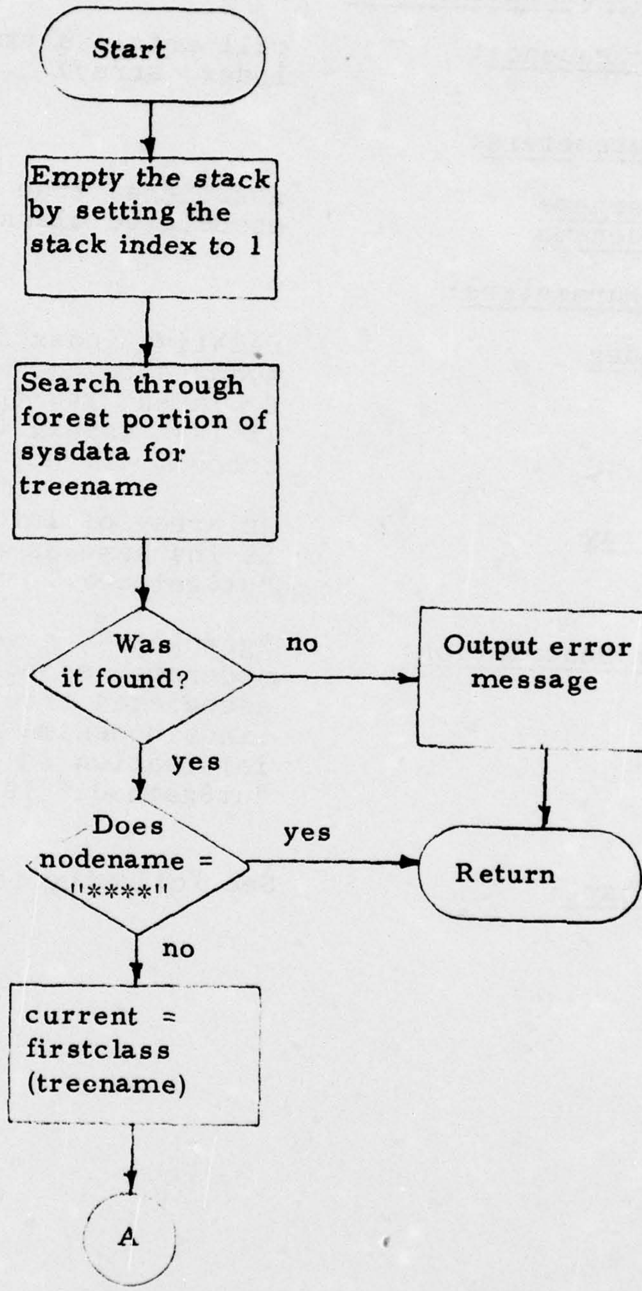
Output Parameters:

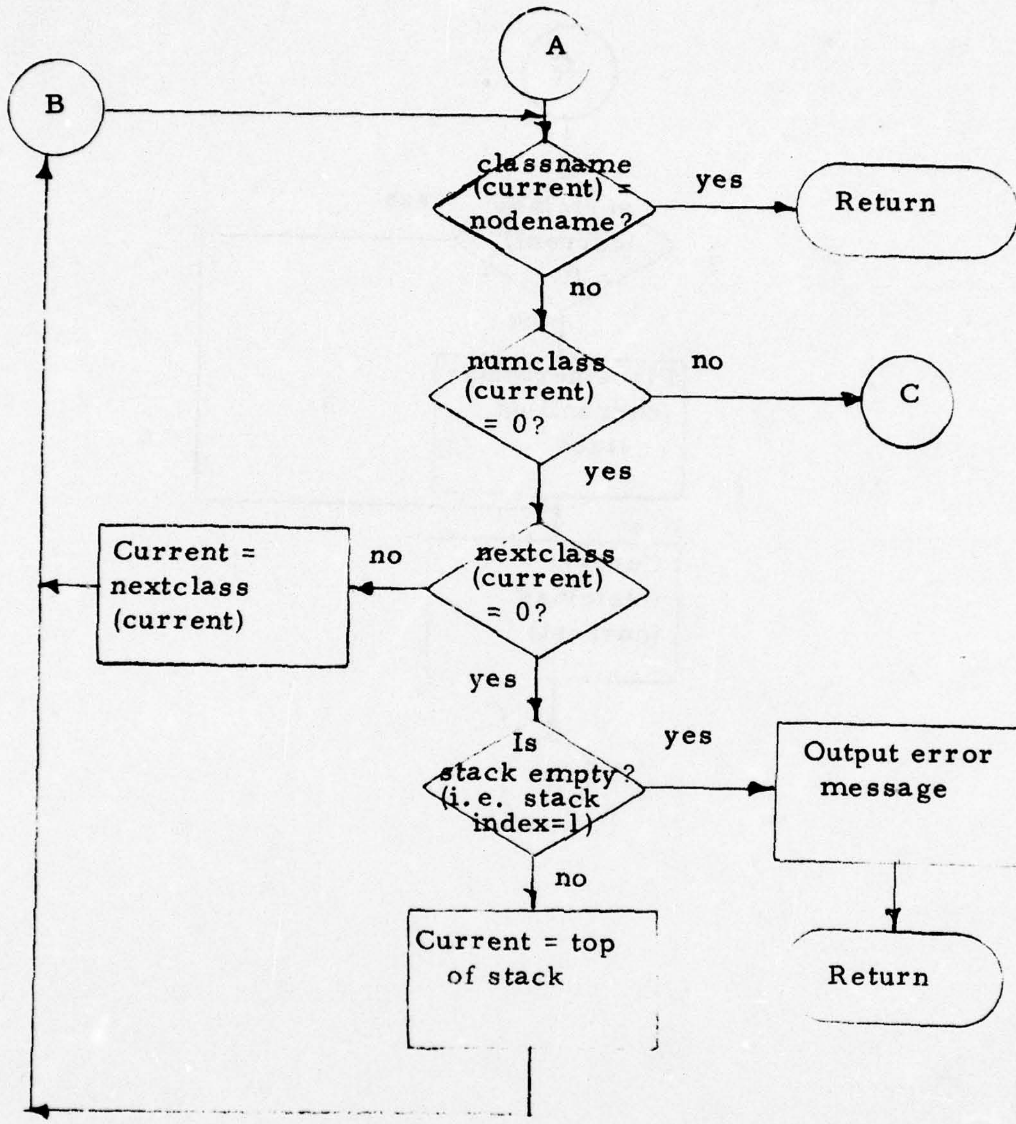
index	relative index from beginning of sysdata to nodename entry. If $0 < \text{index} \leq 144$ then entry is in forest. If $144 < \text{index}$, then entry is in school. If not found, index = -1.
array	an array of information whose data is the same as array of subroutine "ut$getnode."

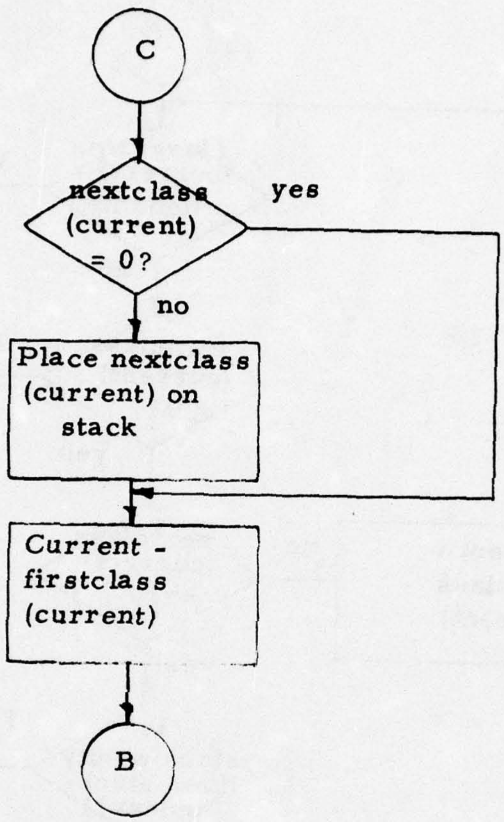
Program Description: "getclass", given a treename and nodename, searches through the associated tree in sysdata until it finds nodename and returns the node information in array. Subroutine "ut$getnode" is used.

Flow Chart: See following page

getclass


Start
Empty the stack by setting the stack index to 1
Search through forest portion of sysdata for treename
Was it found?
no
Output error message
yes
Does nodename = "****"
yes
Return
no
current = firstclass (treename)
A


A
B
classname (current) = nodename?
yes
Return
no
numclass (current) = 0?
no
C
yes
Current = nextclass (current)
no
nextclass (current) = 0?
yes
Is stack empty? (i.e. stack index=1)
yes
Output error message
no
Current = top of stack
Return


C
nextclass
(current)
= 0?
yes
no
Place nextclass
(current) on
stack
Current -
firstclass
(current)
B

Internal Subroutine Name: getclass1

Calling Sequence: call getclass1 (treename, nodename, index, array, sysptr)

Input Parameters:

treename	associated tree name
nodename	associated class name
sysptr	pointer to sysdata

Output Parameters:

index	relative index from beginning of sysdata to nodename entry. If $0 < \text{index} \leq 144$, then entry is in forest. If $144 < \text{index}$, then entry is in school. If not found, index = -1.
array	an array of information whose data are the same as array of subroutine "ut$getnode."

Program Description: "getclass1," given a treename and nodename, searches through the associated tree in sysdata until it finds nodename; it then returns the node information in array. Subroutine "ut$getnode" is used.

Since a pointer to sysdata is passed as a parameter to getclass1, no call is made to hcs_$initiate to determine this pointer, as is done in getclass.

Flow Chart: See getclass.

Internal Subroutine Name: getlabel

Calling Sequence: call getlabel

Input Parameters:

label — fixed (35) external static variable containing the current label

Program Description: getlabel generates the next sequential label for a FORTRAN program

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: getparam

Calling Sequence: **call getparam (ptr, data_type)**

Input Parameters:

ptr	"ptr" is a ptr to the parameter list. This can be obtained by: "call cu$arg_list_ptr(ptr)"
data_type	The type of data that is being inputed as parameters: if "1", data is fixed bin (35) if "2", data is float if "3", data is character string of maximum length 8

Output Parameters:

ptr	"ptr" will point to word 6 of sysdata which is the word where the number of parameters is placed.
data type	"data_type" is set to -1 if any errors occur. Possible errors that could occur are: 1. A character is encountered when trying to input data of data_type 1 or 2 (character other than "." for for type 2), 2. More than 8 characters in a parameter of data_type = 3, 3. Data_type ≠ 1, 2, or 3.

Output File Setting:

sysdata word	contents
6	nparms (number of parameters)
7	parameter(1)
⋮	⋮
6+nparms	parameter (nparms)

Program Description: "getparam" gets the input parameters of a program and places them in words 6 through (6+no. of params,) in 3 possible different formats.

If data_type = 1 each parameter is converted to fixed bin (35) format before placing in sysdata

If data_type = 2 each parameter is converted to floating format before placing in sysdata.

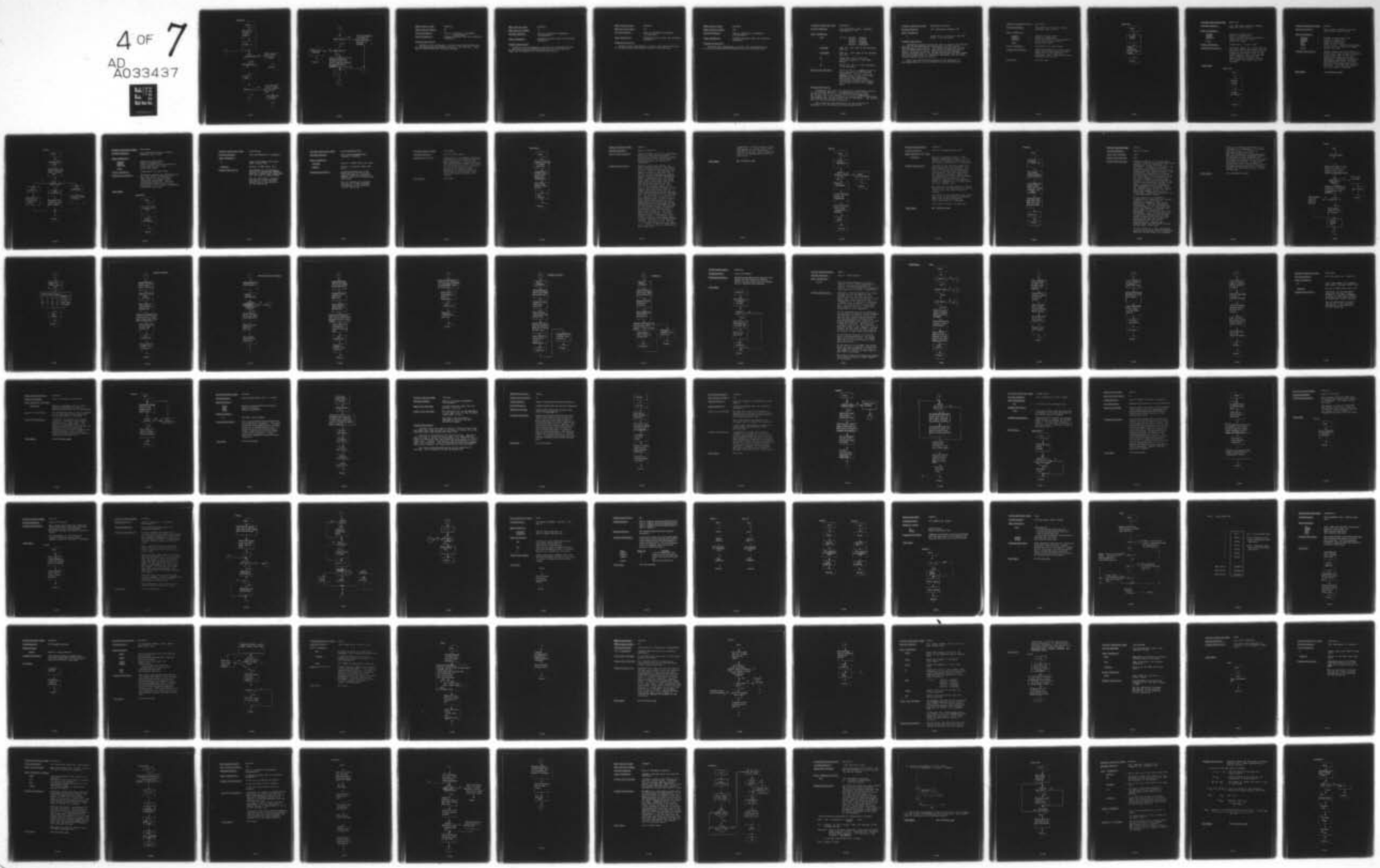
If data_type = 3, each parameter is placed in sysdata in char(8) format. If less than 8 characters are inputed as a parameter, the characters are left justified in each double word in sysdata and pulled to the rights with blanks.

If any errors occur, a data_type of -1 is returned.

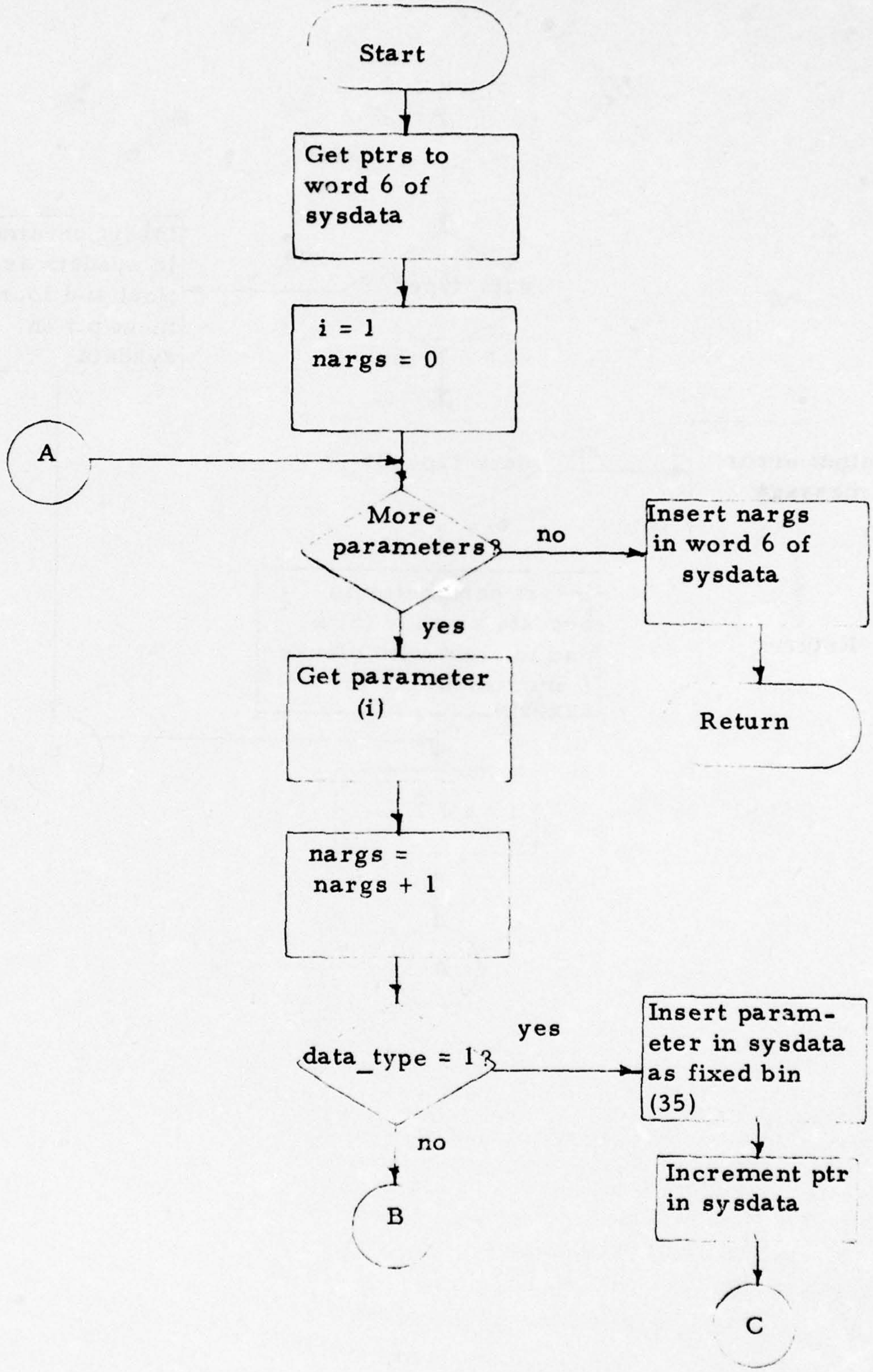
Flow Chart: Next page.

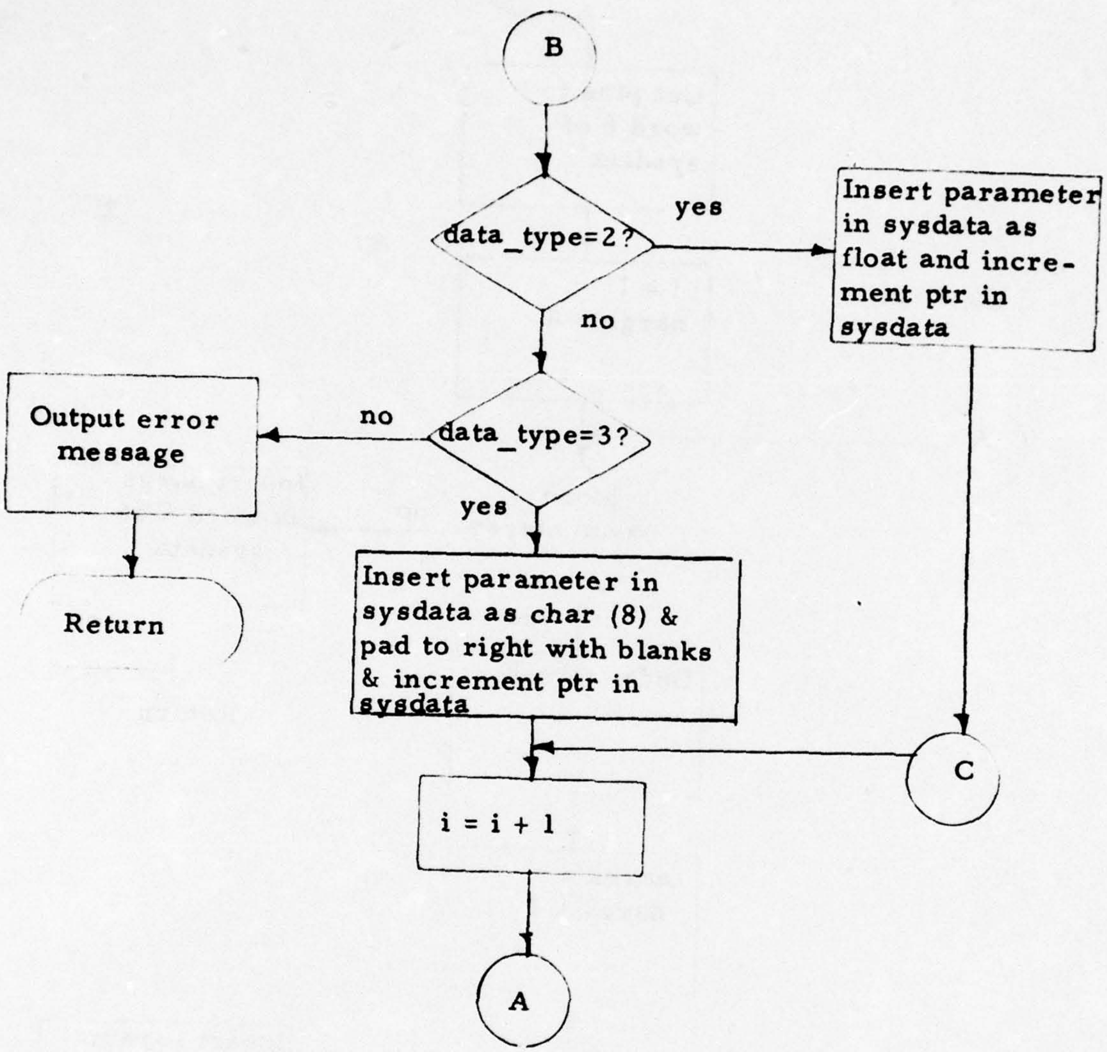
AD-A033 437 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
MULTICS OLPARS OPERATING SYSTEM.(U)
SEP 76 D B CONNELL, K N KLINGBAIL F30602-75-C-0226
UNCLASSIFIED PAR-74-25-B RADC-TR-76-271-VOL-2 NL

4 OF 7
AD A033437



getparam


Start
Get ptrs to word 6 of sysdata
i = 1
nargs = 0
A
More parameters?
no
Insert nargs in word 6 of sysdata
Return
yes
Get parameter (i)
nargs = nargs + 1
data_type = 1?
yes
Insert parameter in sysdata as fixed bin (35)
Increment ptr in sysdata
C
no
B


B
data_type=2?
yes
Insert parameter in sysdata as float and increment ptr in sysdata
no
Output error message
no
data_type=3?
yes
Insert parameter in sysdata as char (8) & pad to right with blanks & increment ptr in sysdata
Return
C
i = i + 1
A

<u>MOOS Function Name</u>: gndv$1dl

<u>MOOS Function Number</u>: 86

<u>Calling Sequence</u>: Type in "gndv$1dl ((treename)) ((classname))"

<u>Input Parameters</u>: Standard optional data set selection parameters

<u>Program Description</u>:

gndv$1dl calls gndv$gendv to project the selected data set on any generalized discriminant vector of the data set. Logic may be created from the resulting display.

<u>MOOS Function Name</u>: gndv$ld2

<u>MOOS Function Number</u>: 72

<u>Calling Sequence</u>: Type in "gndv$ld2 ((treename)) ((classname))"

<u>Input Parameters</u>: Standard optional data set selection parameters

<u>Program Description</u>:

gndv$ld2 calls gndv$gendv to project the selected data set on any two generalized discriminant vectors of the data set. Logic may be created from the resulting display.

MOOS Function Name: gndv$sal

MOOS Function Number: 216

Calling Sequence: Type in "gndv$sal((treename)) ((classname))"

Input Parameters: Standard optional data set selection parameters

Program Description:

gndv$sal calls gndv$gendv to project the selected data set on any generalized discriminant vector of the data set.

MOOS Function Name: gndv$sa2

MOOS Function Number: 204

Calling Sequence: Type in "gndv$sa2 ((treename)) ((classname))"

Input Parameters: Standard optional data set selection parameters

Program Description:

gndv$sa2 calls gndv$gendv to project the selected data set on any two generalized discriminant vectors of the data set.

Internal Subroutine Name: gndv$gendv

Calling Sequence: call gndv$gendv (ptrf, treename, classname, f1, f2)

Input Parameters:

ptrf	(5) ptr	ptrf(1) - sysdata ptrf(2) - scratch ptrf(3) - display ptrf(4) - treename ptrf(5) - mooslogic
treename	char (8)	tree name of the selected data set
classname	char (4)	class name of the selected data set
f1	fixed (35)	Set to zero for structure analysis, 1 for logic design
f2	fixed (35)	Set to 1 for one-space, 2 for two-space

Output File Settings: If f2 is set to 1, csdata is set up for a one-space plot by calling ss$display1. If f2 is set to 2, display is set up for a two-space plot by calling ss$display. Eigenvalues and generalized discriminant vectors are stored in the process directory file eigen_file.

Program Description:

gndv$gendv calculates the generalized discriminant vectors of the selected data set, orthonormalizes these vectors, and stores them in the process directory file eigen_file. The program then calls ss$display (or ss$display1) to initialize the display file (or the csdata file for one-space). The routine exits by calling gndv$gendv_sequence.

For a more detailed description of the operation of gndv$gendv, see the program listing documentation.

Internal Subroutine Name: gndv$gendv_sequence

Calling Sequence: call gndv$gendv_sequence (p)

Input Parameters:

p	fixed (35) p is set to 1 for one-space, 2 for two-space

Program Description:

gndv$gendv_sequence allows the user to project the selected data set on the generalized discriminant vectors without recalculating these vectors. The program displays an ordered list of eigenvalues from the eigen_file and asks the user to select the one(s) he wants. The generalized discriminant vectors corresponding to the chosen eigenvalues are then loaded into the display file (or csdata for one-space) and the appropriate display routine is called.

For a more detailed description of the operation of gndv$gendv_sequence, see the program listing documentation.

Internal Subroutine Name: gpboolean

Calling Sequence: call gpboolean (critptr, numdim, numbound, lnb, ifile)

Input Parameters:

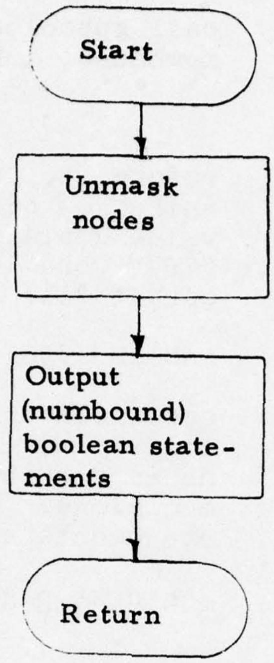
critptr - pointer to logic block
numdim - number of dimensions
numbound - value contained in third quarter of first word in logic block
ifile - output file name

Output Parameter: lnb-array(3) of logic nodes

Program Description: - gpboolean unmasks the second word of logic block, extracting the logic nodes for the true and false decisions. Next (numbound) boolean statements are output.

Flow Chart: See next page

gpboolean


Start
Unmask nodes
Output (numbound) boolean statements
Return

Internal Subroutine Name: gpdiscrim

Calling Sequence: call gpdiscrim (critptr, numdim, numbound, lnb, ifile)

Input Parameters:

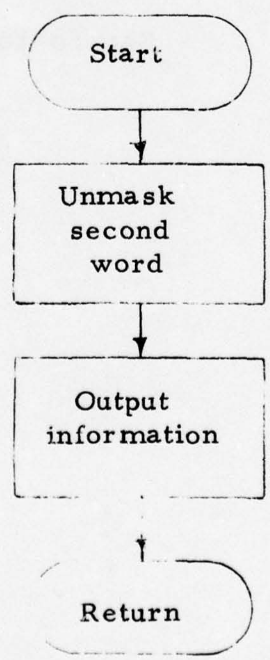
- critptr - pointer to logic block
- numdim - number of dimensions
- numbound - value contained in third quarter of first word in logic block
- ifile - output file name

Output Parameter: lnb-array(3) of logic nodes

Program Description: gpdiscrim unmasks the second word of the logic block, extracting the logic nodes for the convex sides of the (numbound) boundaries, and for the excess region. Next, the discriminant coefficient of each line segment and the threshold for the line segment are output for each boundary.

Flow Chart:

gpdiscrim



Internal Subroutine Name: gplogic

Calling Sequence: call gplogic (fileptr, optfile, numdim, dex, nodes, ifile)

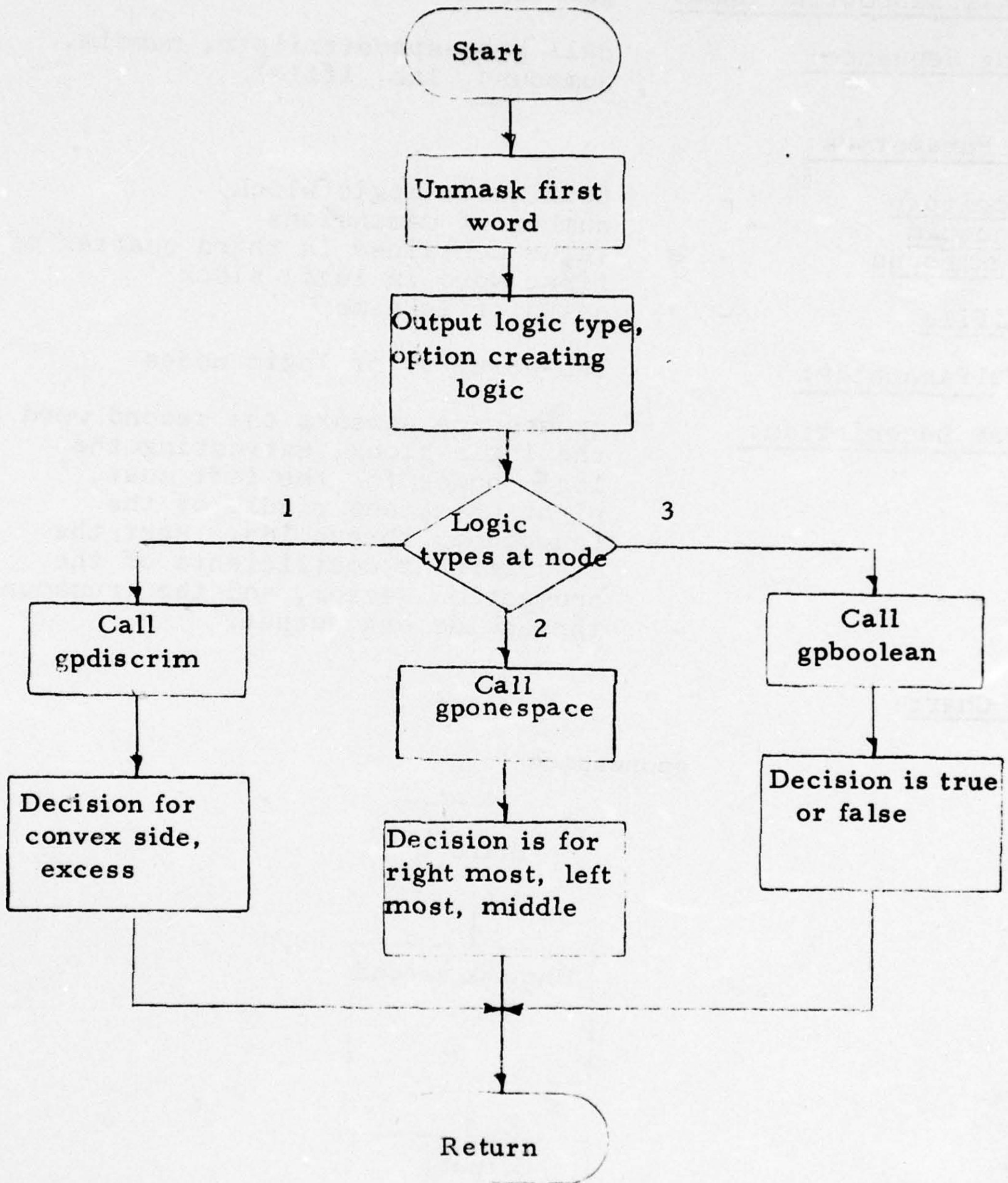
Input Parameters:

fileptr	-	pointer to moos logic file
optfile	-	pointer to option_file
numdim	-	number of dimensions
dex	-	index to logic block
nodes	-	array (72) of 4-character node names at node ifile - output file name
ifile	-	output file name

Program Description: gplogic unmasks the first word in a group logic block, formats the output of the logic type and the option creating the logic, calls the appropriate subroutine (gpdiscrim, gponespace, gpboolean) to output the appropriate values, and then points out the logic node to go to, depending on the decision made at the node, before returning.

Flow Chart: **See following page**

gplogic



Start
Unmask first word
Output logic type, option creating logic
Logic types at node
1
2
3
Call gpdiscrim
Call gponespace
Call gpboolean
Decision for convex side, excess
Decision is for right most, left most, middle
Decision is true or false
Return

Internal Subroutine Name: gponespace

Calling Sequence: call gponespace(critptr, numdim, numbound, lnb, ifile)

Input Parameters:

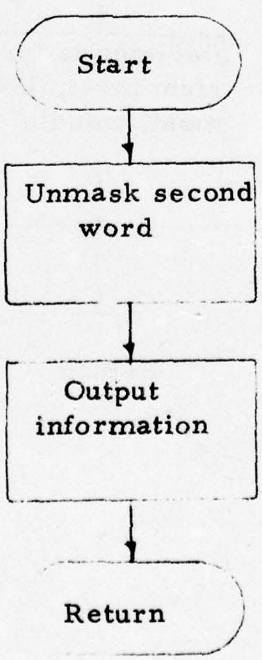
- critptr - pointer to logic block
- numdim - number of dimensions
- numbound - value contained in third quarter of first word in logic block
- ifile - output file name

Output Parameter: lnb-array(3) of logic nodes

Program Description: gponespace unmasks the second word of the logic block, extracting the logic nodes for the left most, right most, and middle of the (numbound) thresholds. Next the discriminant coefficients of the projection vector, and the (numbound) thresholds are output.

Flow Chart:

gponespace



Internal Subroutine Name: grouprogram

Calling Sequence: call grouprogram (k, logicptr)

Input Parameters:

k — logic node number with group logic (fixed (35))

logicptr — pointer to MOOS logic file

Program Description: grouprogram is the executive routine for generating FORTRAN code for a group logic node under the "fortlogc" option of MOOS

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: grouprogram$gboolean

Calling Sequence: call grouprogram$gboolean (logicptr, nodeptr)

Input Parameters:

logicptr — pointer to MOOS logic file (ptr)

nodeptr — pointer to boolean logic node (ptr)

Program Description: grouprogram$gboolean is the subroutine in the "fortlogc" option of MOOS that generates FORTRAN code for a boolean group logic node

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

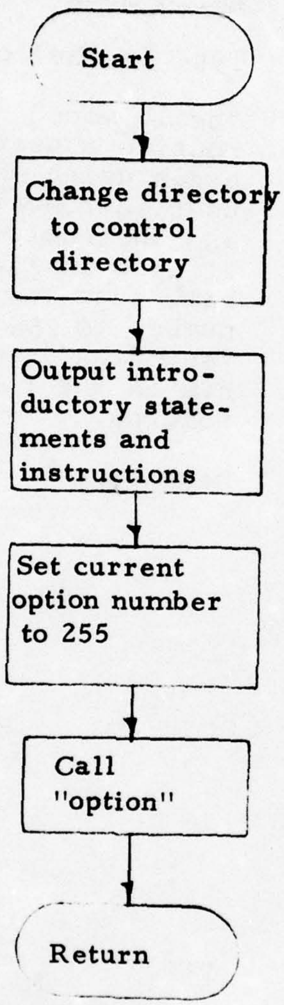
Utility Function Name: hello_moos

Calling Sequence: Type in "hello_moos"

Program Description: "hello_moos" is normally the first routine a user uses when he wants to begin using moos. It gives the user introductory remarks about moos and instructs him on how to use moos. It also initializes all files for him, sets the current option number to 255 (option number for "anything"), calls "option" puts him in the right directory and returns.

Flow Chart: next page

hello_moos


Start
Change directory to control directory
Output introductory statements and instructions
Set current option number to 255
Call "option"
Return

<u>Utility Function Name:</u> hgprint

<u>Calling Sequence:</u> type in "hgprint"

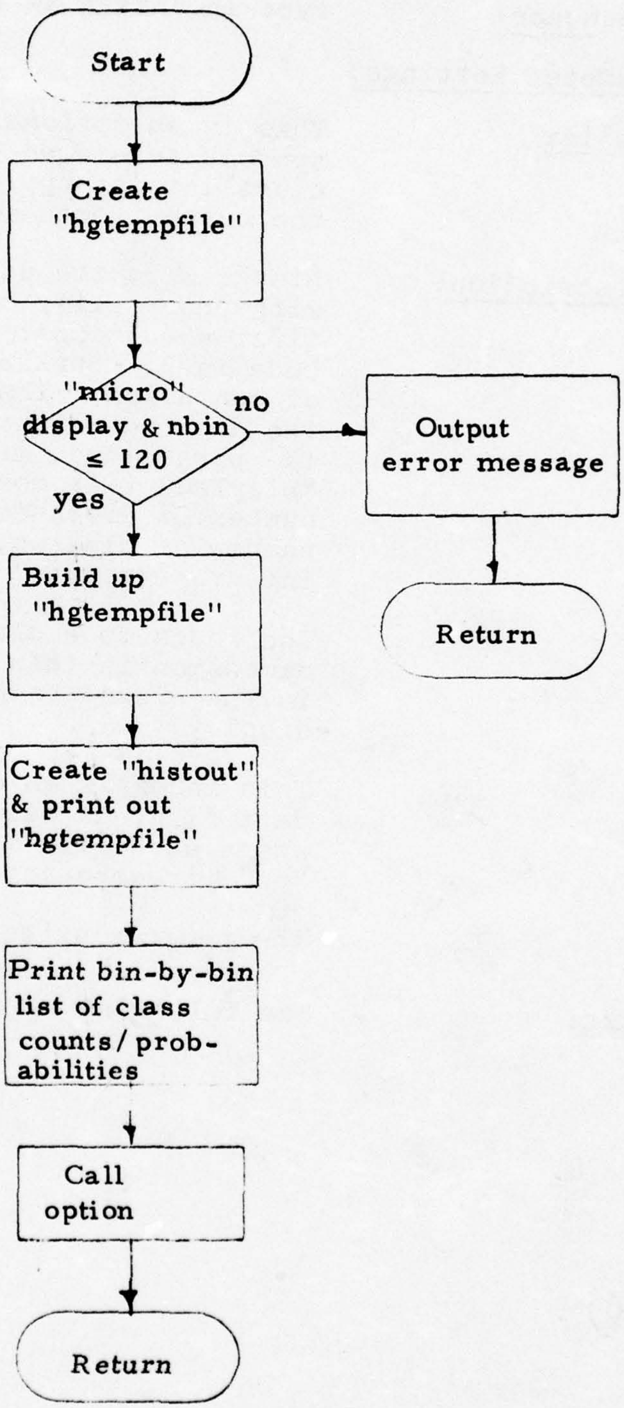
<u>Output File Settings:</u> hgprint creates two files: hgtempfile" in the process directory which mirrors the current one-space display and "histout" in the user's home directory, which is similar to hgtempfile".

<u>Program Description:</u> hgprint outputs the present one-space "micro" display to the printer. It first checks if the current display is a "micro" view and that the number of bins is less than 120. If not, an error message is printed. hgprint then constructs "hgtempfile" which is number of bins x 30 words long. Each consecutive nbin words represents a horizontal line of the current display and each of the nbin words is either a "b̸", "*", "+", or the display symbol of the class. Each column is headed by the appropriate class symbol and is filled with asterisks unless another class distribution falls in that same bin. In that case, a "+" is printed as a header if both classes have equal distribution, or the different class symbols are printed at the appropriate positions in the column. Upon completion of "hgtempfile", the MULTICS system routine "file_output" is called with parameter "histout". This creates the file "histout" which will contain all the information that is printed. A column spacing factor is determined by the number of bins. If nbin<20, then 3 print positions are skipped between columns. If nbin $\geq$ 20 and nbin $\leq$ 30, then the skip factor is 2. If bin · 30 and $\leq$ 50 then the skip factor is 1, else the output is consecutive print positions.

"hgtempfile" is then printed a line, nbin words, at a time. Other output information is the current moos function, data set, and a bin-by-bin list of the probabilities/count of each class.

Flow Chart: See following page

hgprint


Start
Create "hgtempfile"
"micro" display & nbin ≤ 120
no
yes
Output error message
Return
Build up "hgtempfile"
Create "histout" & print out "hgtempfile"
Print bin-by-bin list of class counts/probabilities
Call option
Return

User Utility Function: histgram

Calling Sequence: type in "histgram [classlist]"

Input Parameter Settings:

classlist: This is an optional list of class symbols separated by blanks. If no classlist is supplied, all classes of the current data set will be displayed.

Program Description: histgram is the display program used with probability of confusion. It first checks that the system display code is 2, representing probability of confusion. The user then enters the desired measurement number, and the parameters, if any, are determined. "Display" file contents such as number of dimensions, tree character, number of classes, etc. is then copied into the "csdata" file.

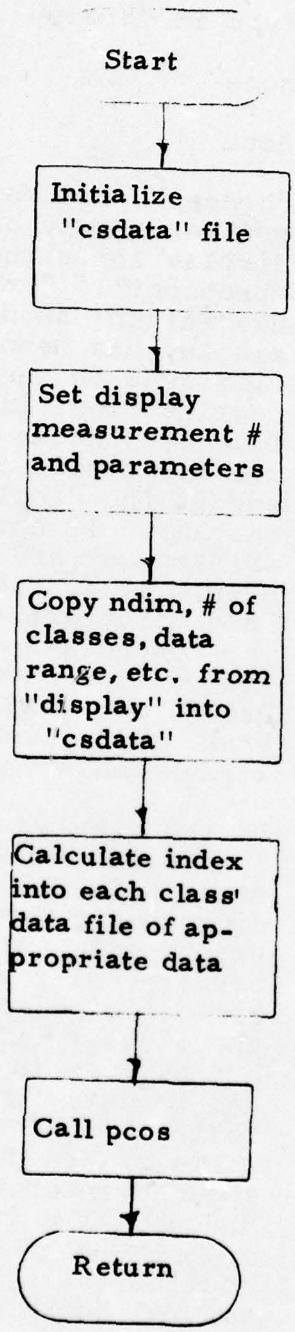
The index to a class' data for a given dimension is then determined and stored in the class' four-word section of display.

This index is an offset into the class' data file, as calculated by probconf, where each file is denoted by "pc" || treecharacter || classname.

The program exits by calling pcos.

Flow Chart: See following page

histgram


Start
Initialize "csdata" file
Set display measurement # and parameters
Copy ndim, # of classes, data range, etc. from "display" into "csdata"
Calculate index into each class' data file of appropriate data
Call pcos
Return

Utility Function Name: hrdcpy

Calling Sequence: Type in "hrdcpy"

Input File Setting: none

Output File Setting: none

Program Description: "hrdcpy" produces on the high speed printer a copy of any desired rank order display for either "dscrmeas" or "probconf". "hrdcpy" is called by the user after the desired rank order display has been called and the results displayed on the terminal screen. The routine first initiates the display file and checks the first nine bits of the first word in the display file to see if the display is a rank order display. If not, an error message is printed out and control returns to the user. If so, then the sortorder is read to determine if the display is for "dscrmeas" or "probconf". Next, the routine checks to see the type of rank order display called, ie., "rnk$oall", "rnk$bcls", "rnk$bycp", or a rank measurement display.

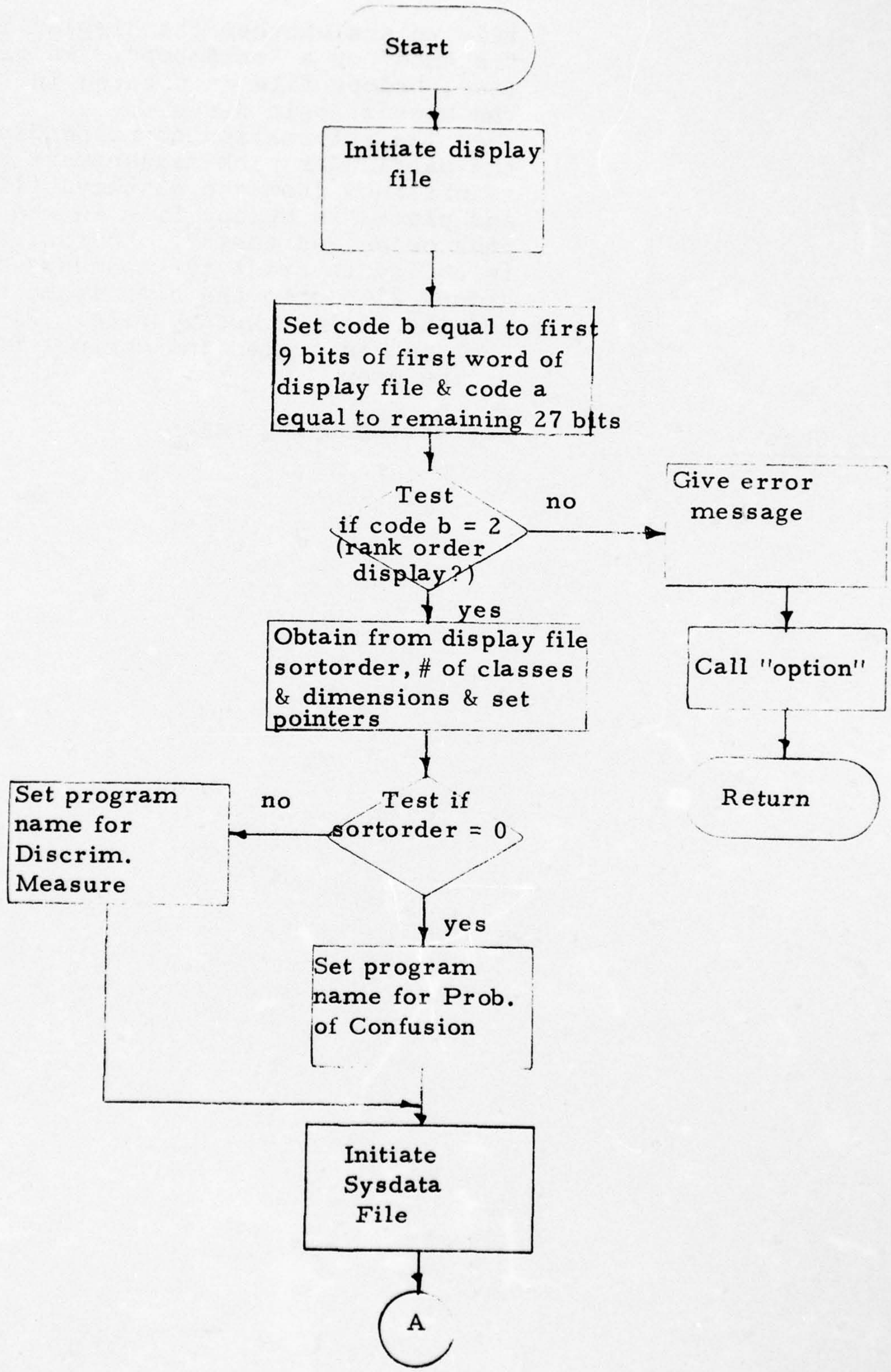
If the display is a "rnk$oall", "rnk$bcls", or "rnk$bycp" the routine creates in the user's login directory a temporary file called hrdcpy_file. Then the information corresponding to the particular rank order file display is obtained from the display file and placed in hrdcpy_file in the correct rank order and format. Then the routine calls "output_file" which prints the contents of hrdcpy_file onto the high speed printer and then deletes hrdcopy_file from the user's login directory. Finally the routine calls "option" which returns control back to the user.

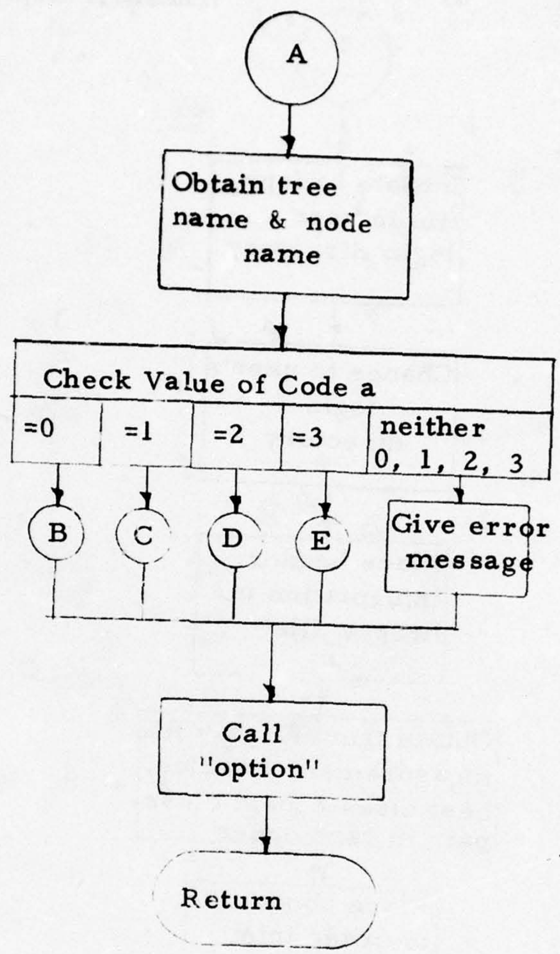
If the display is a rank measurement display the displaya file is initiated. Then the routine checks the displaya

file to see whether the display is a "rnk$mbc" or a "rnk$mbcp". In either case, hrdcpy_file is created in the user's login directory.
Then the information corresponding to the particular rank measurement display is obtained from the displaya file and placed in hrdcpy_file in the correct rank order and format. "output_file" is called to print the contents of hrdcpy_file onto the high speed printer and then delete hrdcpy_file. Finally "option" is called and control returns to the user.

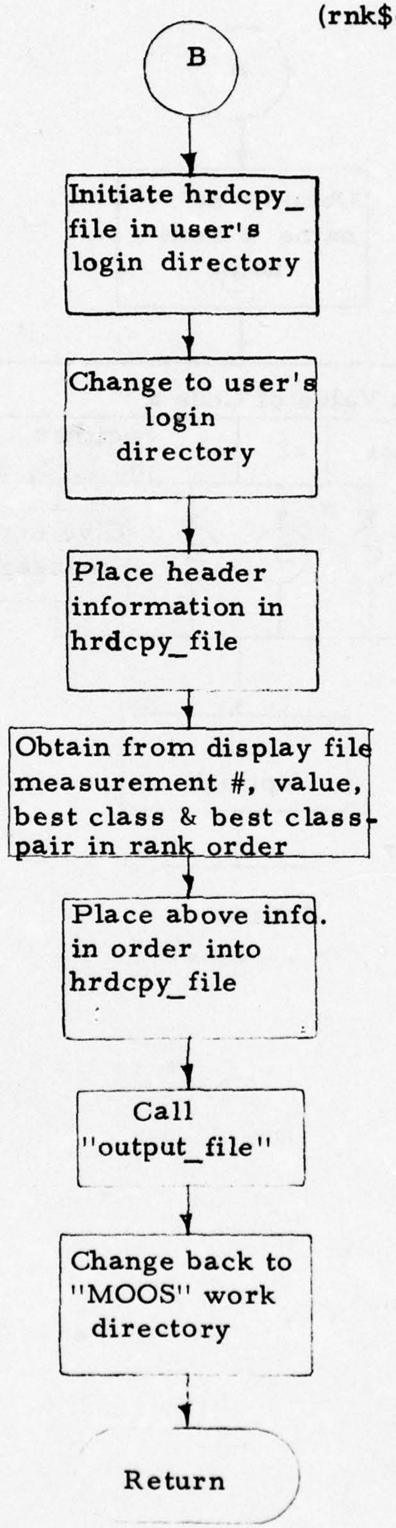
Flow Chart: See accompanying pages

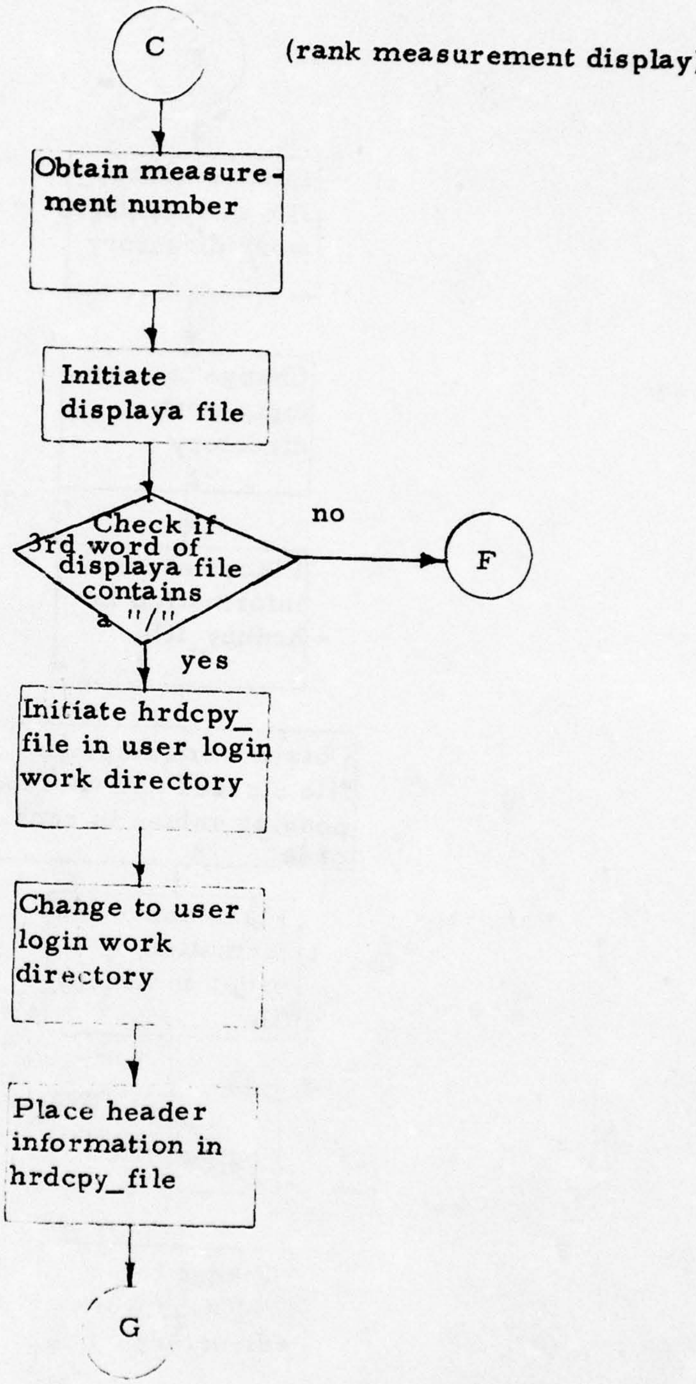
hrdcpy


Start
Initiate display file
Set code b equal to first 9 bits of first word of display file & code a equal to remaining 27 bits
Test if code b = 2 (rank order display?)
no
Give error message
Call "option"
Return
yes
Obtain from display file sortorder, # of classes & dimensions & set pointers
Test if sortorder = 0
no
Set program name for Discrim. Measure
yes
Set program name for Prob. of Confusion
Initiate Sysdata File
A

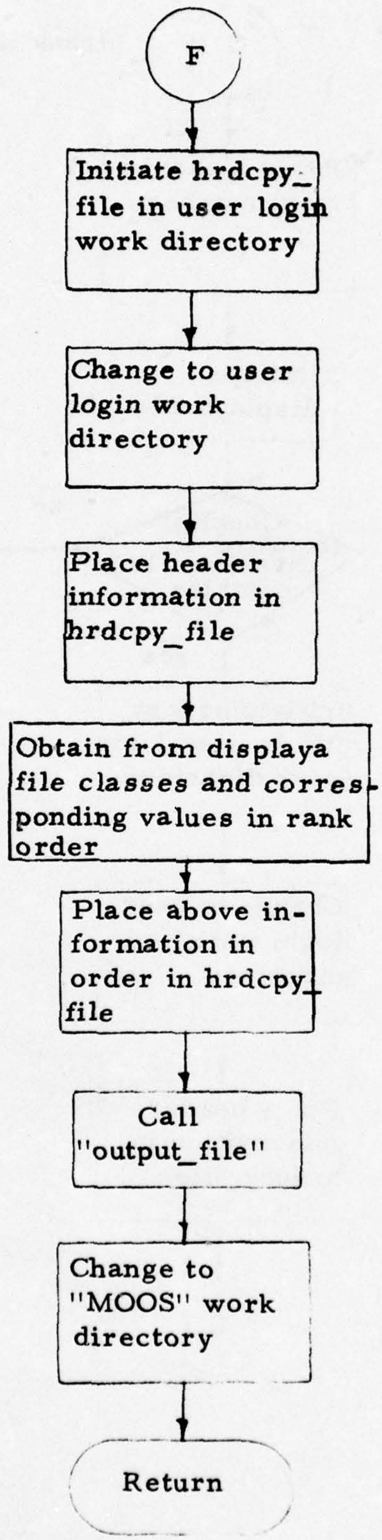

A
Obtain tree name & node name
Check Value of Code a
=0
=1
=2
=3
neither 0, 1, 2, 3
B
C
D
E
Give error message
Call "option"
Return

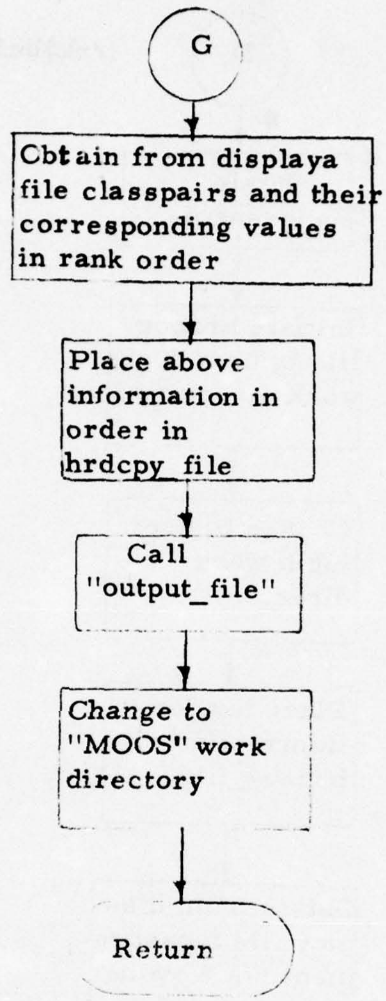
(rnk$oall display)

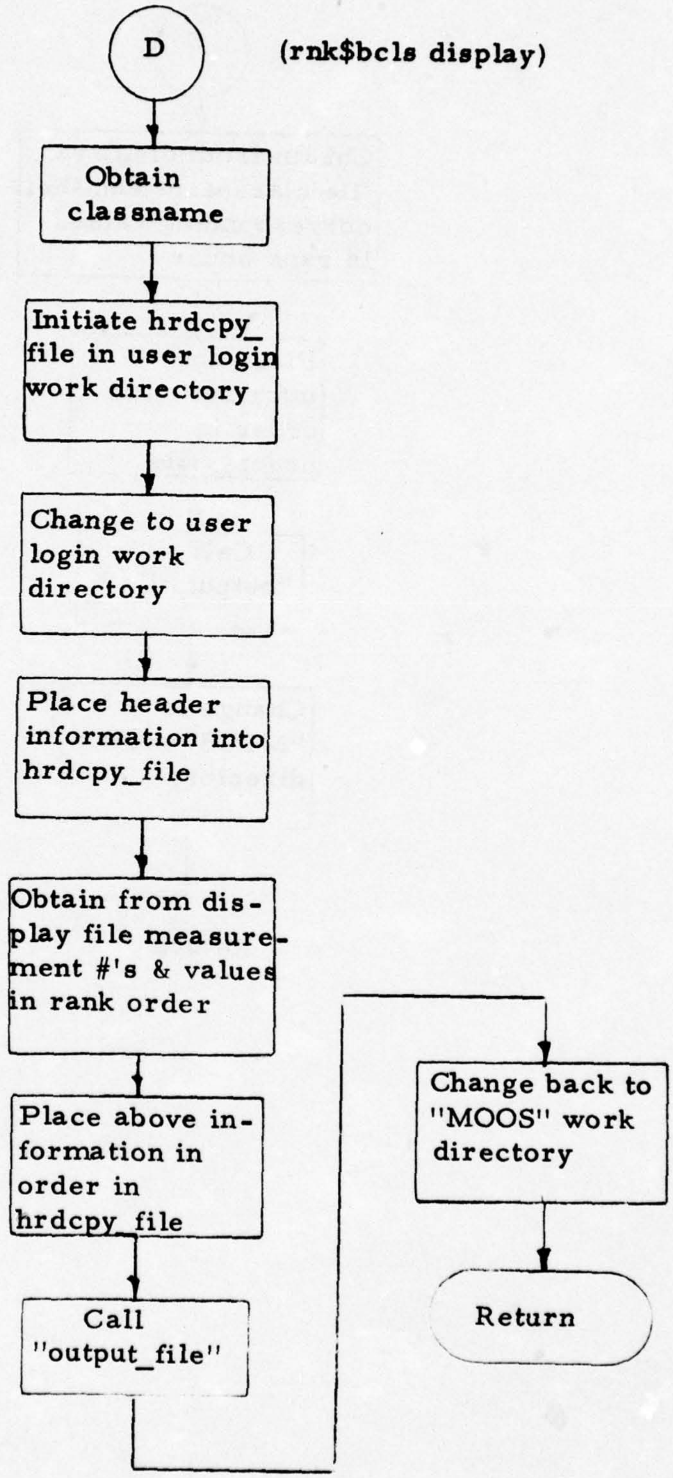


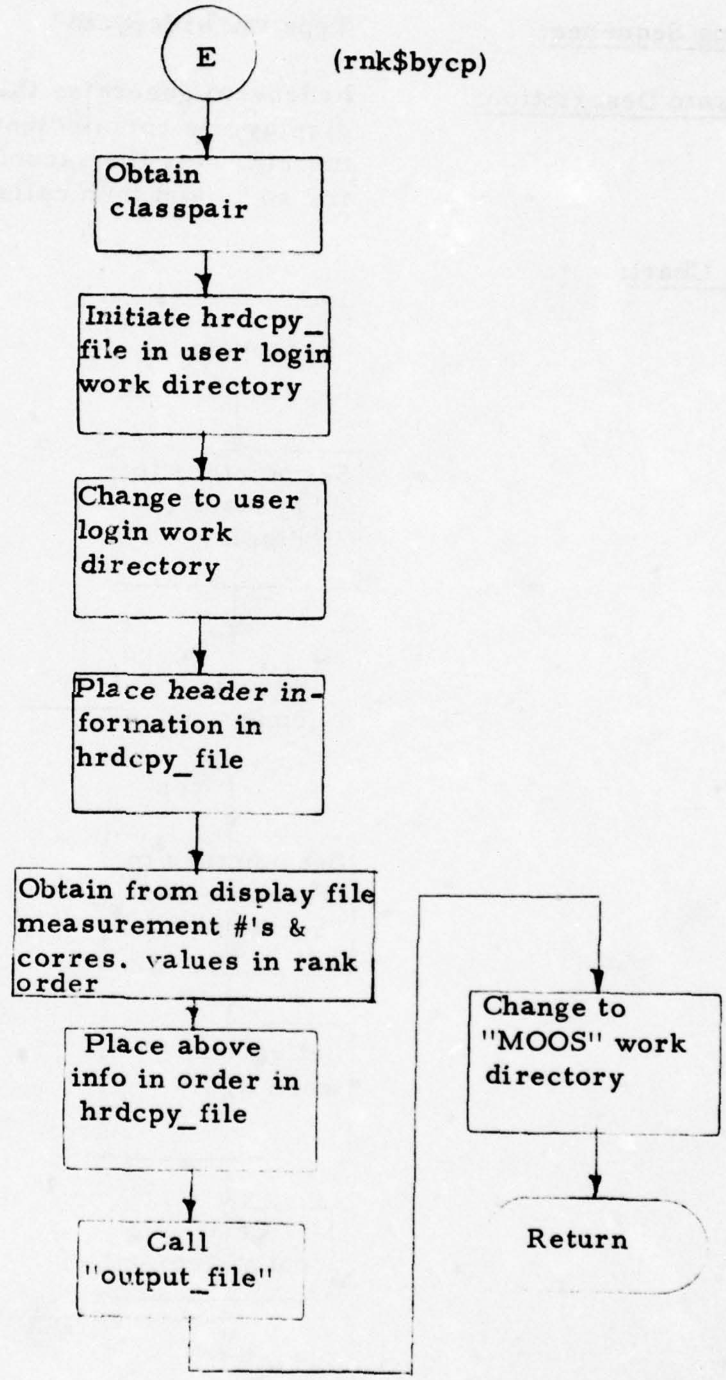


C
(rank measurement display)
Obtain measurement number
Initiate displaya file
Check if 3rd word of displaya file contains a "/"
no
F
yes
Initiate hrdcpy_ file in user login work directory
Change to user login work directory
Place header information in hrdcpy_file
G


F
Initiate hrdcpy_ file in user login work directory
Change to user login work directory
Place header information in hrdcpy_file
Obtain from displaya file classes and corresponding values in rank order
Place above information in order in hrdcpy_file
Call "output_file"
Change to "MOOS" work directory
Return


G
Cbtain from displaya file classpairs and their corresponding values in rank order
Place above information in order in hrdcpy_file
Call "output_file"
Change to "MOOS" work directory
Return


D
(rnk$bcls display)
Obtain classname
Initiate hrdcpy_ file in user login work directory
Change to user login work directory
Place header information into hrdcpy_file
Obtain from display file measurement #'s & values in rank order
Place above information in order in hrdcpy_file
Call "output_file"
Change back to "MOOS" work directory
Return

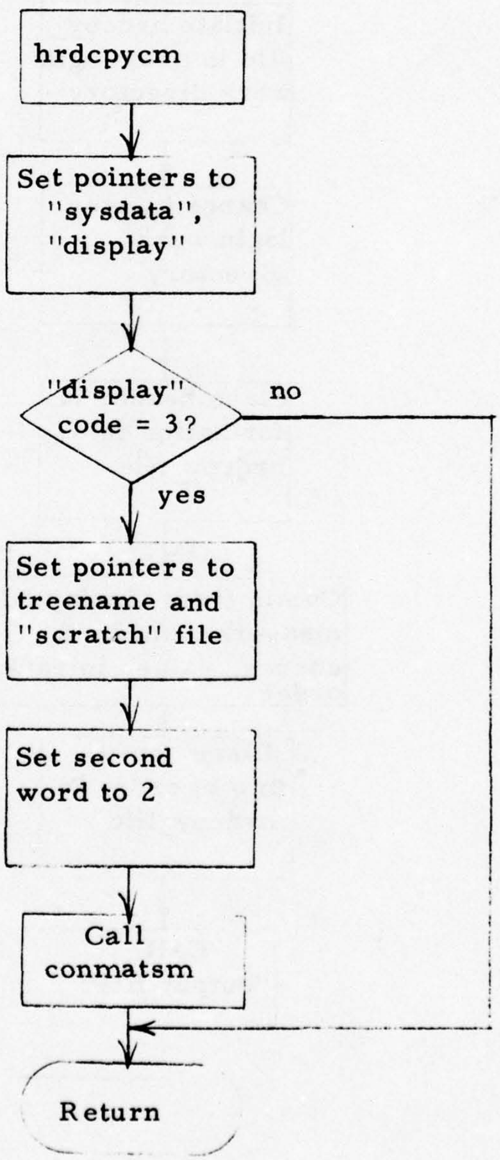

E
(rnk$bycp)
Obtain classpair
Initiate hrdcpy_file in user login work directory
Change to user login work directory
Place header information in hrdcpy_file
Obtain from display file measurement #'s & corres. values in rank order
Place above info in order in hrdcpy_file
Call "output_file"
Change to "MOOS" work directory
Return

Utility Function Name: hrdcpycm

Calling Sequence: Type in "hrdcpycm"

Program Description: hrdcpycm generates the four pointers if the display file code indicates a confusion matrix, sets the second word of the display file to 2, and then calls conmatsm.

Flow Chart:



Utility Function Name: index

Calling Sequence: type in "index [parm]"

Input Parameters:

[parm] This optional parameter is either "count" or "id", this determines the type of information presented on console. In the absence of a parameter, index defaults to "count" information.

Program Description: Initially, "index" examines the system display code to determine if the current display is a one or a two-space plot. If the user desires a two-space index, then word D6 of the "display" file is checked to determine if there exist a cluster or a scatter plot. In the case of a cluster plot, the crosshairs are used to indicate which grid is to be examined.

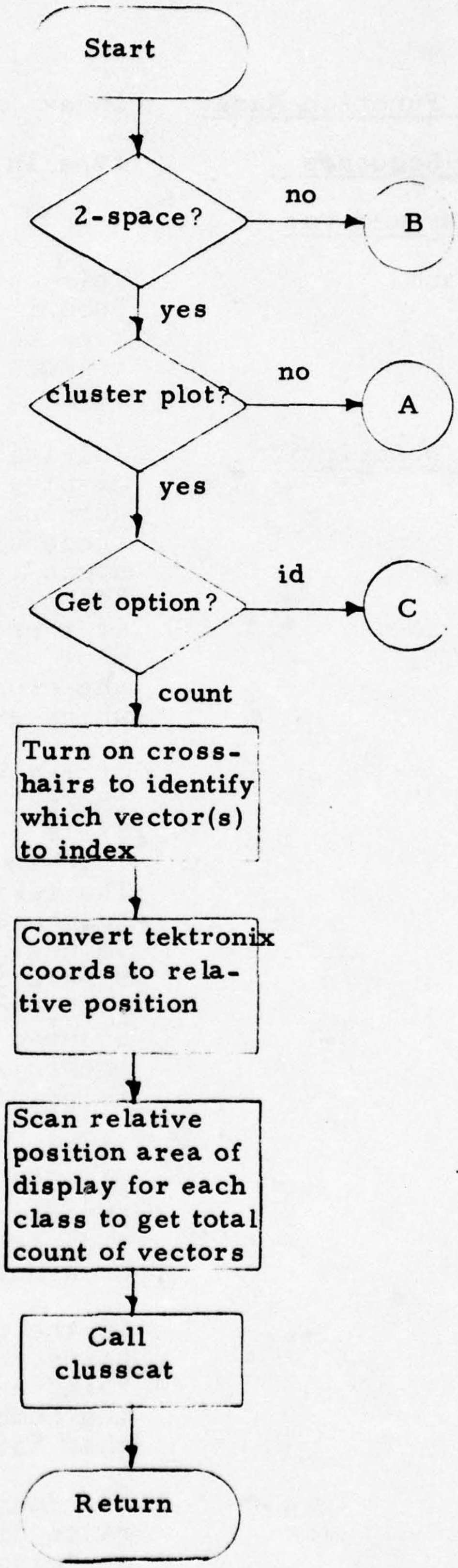
If the information is to be placed on the printer, the MULTICS system program, "file_output" is used to create a "P" file in the current working directory. The tektronix points returned from multeks$read_xhair are converted to data values and these are converted into a relative position. The "relative position" area of the "display" file is scanned and the total number of vectors present in that grid for each class is is printed to the correct destination.

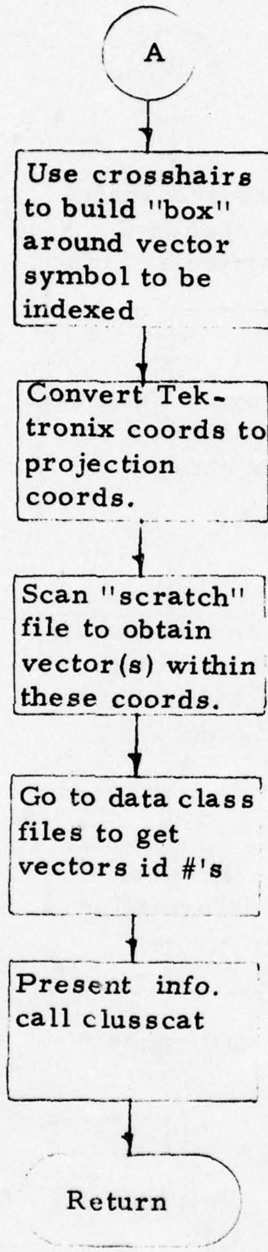
If the current display is a scatter plot, "multeks$read_xhair" is called twice to outline an area. The class and vector id of each vector in this area are printed.

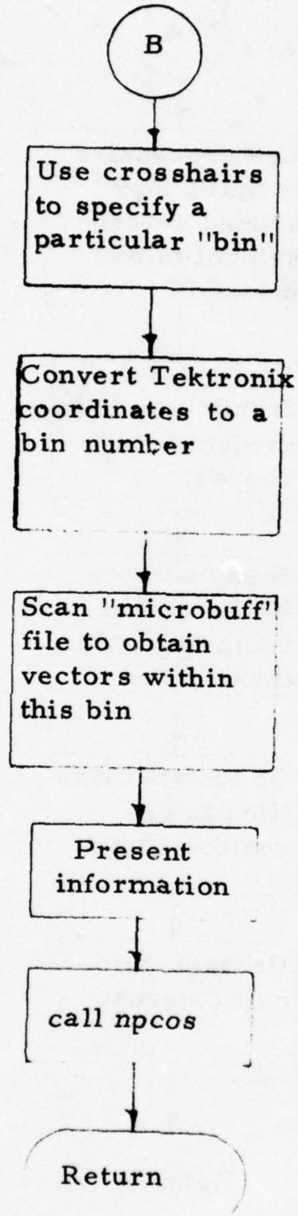
In the case of a histogram, the crosshairs are used to specify a particular "bin" and the probability or count of the number of vectors of each class in that "bin" is printed.

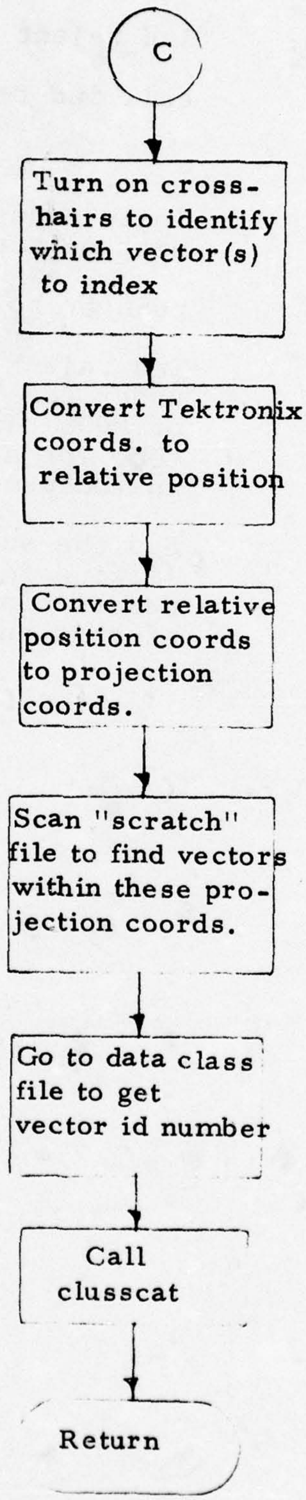
The routine exits by calling the appropriate display routine, either "npcos" or "clusscat".

Flow Chart: index




A
Use crosshairs to build "box" around vector symbol to be indexed
Convert Tektronix coords to projection coords.
Scan "scratch" file to obtain vector(s) within these coords.
Go to data class files to get vectors id #'s
Present info. call clusscat
Return


B
Use crosshairs to specify a particular "bin"
Convert Tektronix coordinates to a bin number
Scan "microbuff" file to obtain vectors within this bin
Present information
call npcos
Return


C
Turn on cross-hairs to identify which vector(s) to index
Convert Tektronix coords. to relative position
Convert relative position coords to projection coords.
Scan "scratch" file to find vectors within these projection coords.
Go to data class file to get vector id number
Call clusscat
Return

Internal Subroutine Name: ind_reject

Calling Sequence: call ind_reject (ii, logicptr)

Input Parameters:

ii — logic node number with independent reject strategy (fixed (35))

logicptr — pointer to MOOS logic file (ptr)

Program Description: ind_reject is the subroutine under the "fortlogc" option of MOOS that generates FORTRAN code for a logic node which has an independent reject strategy

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

User Utility Function: intensfy

Calling Sequence: type in "intensfy [classlist]"

Input Parameter Settings:

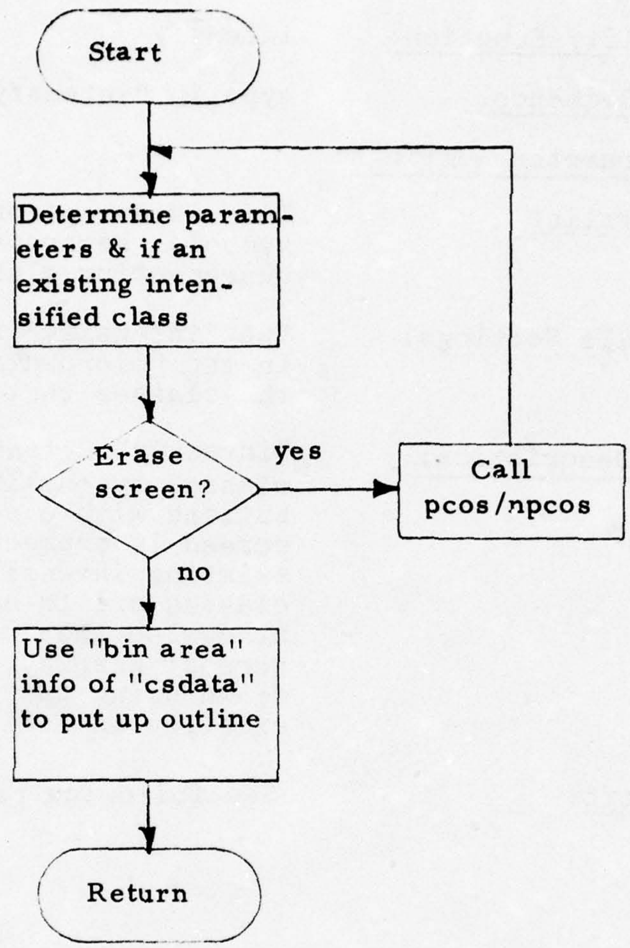
classlist — This is an optional list of class symbols, separated by blanks, representing classes to be intensified.

Output File Settings: The "intensify" flag is set or cleared in the "microbuff" file reflecting the classes that are affected.

Program Description: "intensfy" intensifies the desired classes by outlining their distributions with a solid curve. The screen is erased when there is an existing intensified class and not all classes are to be intensified, otherwise the outline is drawn on the current display. The outline is drawn using the "binned" information existing in "csdata" file.

Flow Chart: See following page

intensfy


Start
Determine param-
eters & if an
existing inten-
sified class
Erase
screen?
yes
no
Call
pcos/npcos
Use "bin area"
info of "csdata"
to put up outline
Return

Internal Subroutine Name: invertmat

Calling Sequence: call invertmat (ptrwl, ptric, w, mnsn)

Input Parameters:

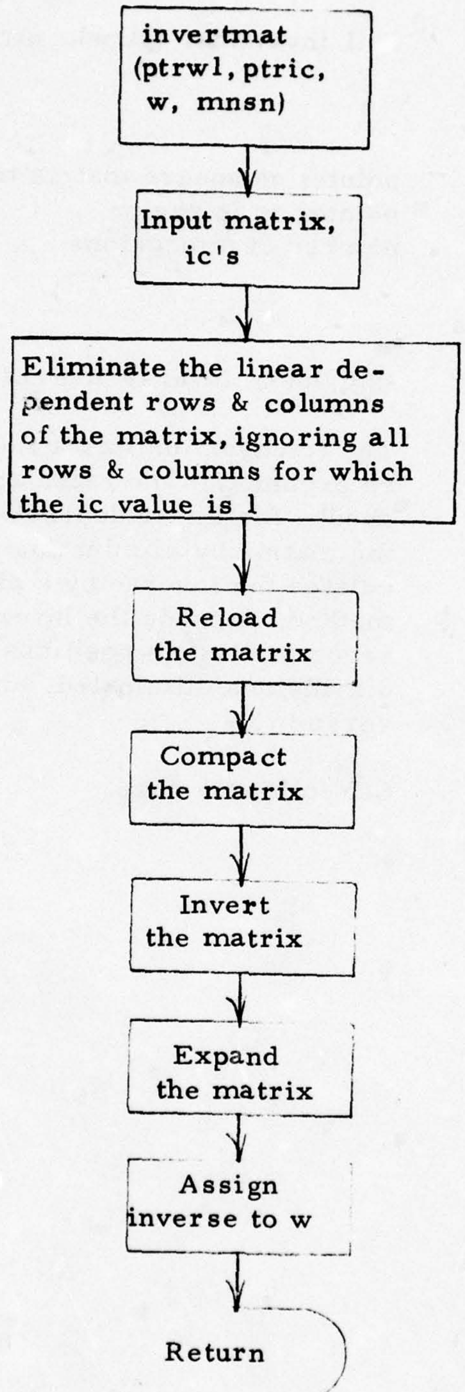
ptrwl	-	pointer to square matrix to be inverted.
ptric	-	pointer to ic region
mnsn	-	number of dimensions

Output Parameter:

w	-	(100, 100), inverse matrix

Program Description: The ic region indicates those dimensions to be excluded in the calculations. The routine checks for linear dependent rows, compacts the matrix by eliminating these rows, calculates the inverse by a pivotal in place method, expands the inverse by inserting zeros in all (i, j) positions, i, j those dimensions eliminated, and returns the inverse in w .

Flow Chart: See following page.


invertmat (ptrwl, ptric, w, mnsn)
Input matrix, ic's
Eliminate the linear dependent rows & columns of the matrix, ignoring all rows & columns for which the ic value is 1
Reload the matrix
Compact the matrix
Invert the matrix
Expand the matrix
Assign inverse to w
Return

Utility Function Name: latclogc

Calling Sequence: Type in "latclogc((treename)) ((classname))"

Input File Settings: A logic file must exist for the selected data set.

Output File Settings: The low-order bit of the SN4 entry in the logic file for the "lattice logic node" is set.

The logic tree structure is modified to reflect the user-specified changes.

Program Description:

latclogc allows the user to modify a selected logic tree such that more than one path may be taken to arrive at a user-specified logic node within the logic tree.

latclogc is divided into two major sections. Section I asks the user to enter the logic nodes to be connected and checks to make sure that the connection will result in a valid logic tree. Section II carries out the modification to the logic tree structure. The routine ends by printing a message which states whether the logic modification was successful.

For a more detailed description of the operation of latclogc, see the program listing documentation.

MOOS Function Name: linglogc

MOOS Function Number: 62

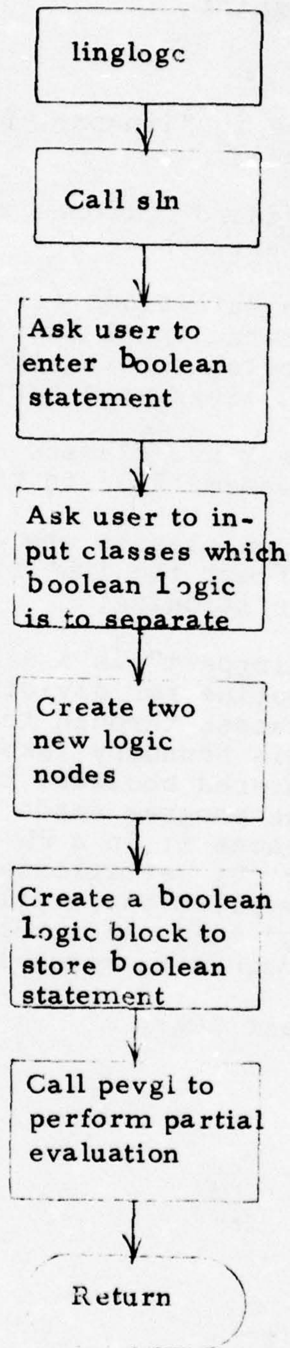
Calling Sequence: Type in "linglogc [(treename)] [(nodename)] "

Input Parameters: Standard optional data set selection parameters

Output File Settings: linglogc adds a boolean or linguistic logic node to the mooslogic file.

Program Description: linglogc calls internal subroutine sln which selects a logic node to be the current logic node. The user is then asked to input a boolean statement for partition logic and the classes which this logic is to separate. Two new nodes are then created in the node part of the structure part of the mooslogic file and a "boolean logic block" is created to store the entered statement. The maximum length of the entered statement is 132 characters. The program calls pevgl to perform a partial evaluation of the boolean logic and exits.

Flow Chart: See following page.


linglogc
Call sln
Ask user to enter boolean statement
Ask user to input classes which boolean logic is to separate
Create two new logic nodes
Create a boolean logic block to store boolean statement
Call pevgl to perform partial evaluation
Return

Moos Function Name: lingpart

Moos Function Number: 198

Calling Sequence: Type in "lingpart [(treename)] [(nodename)]"

Input Parameters: Standard optional data set selection parameters

Output File Settings: The data class file for nodename is deleted and 2 new data class files are created for the true and false side of the linguistic partition.

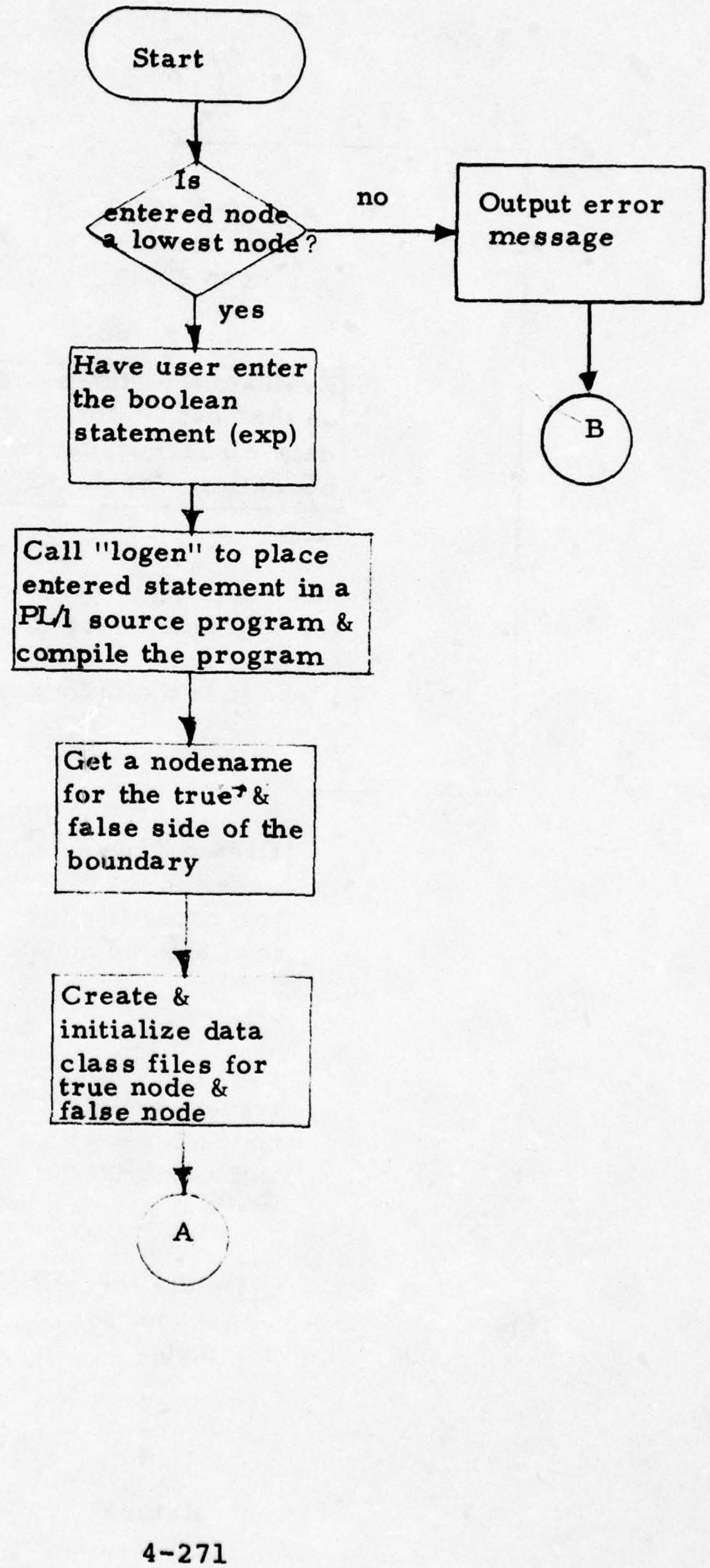
The 2 new classes are added to the treename file to reflect the partition.

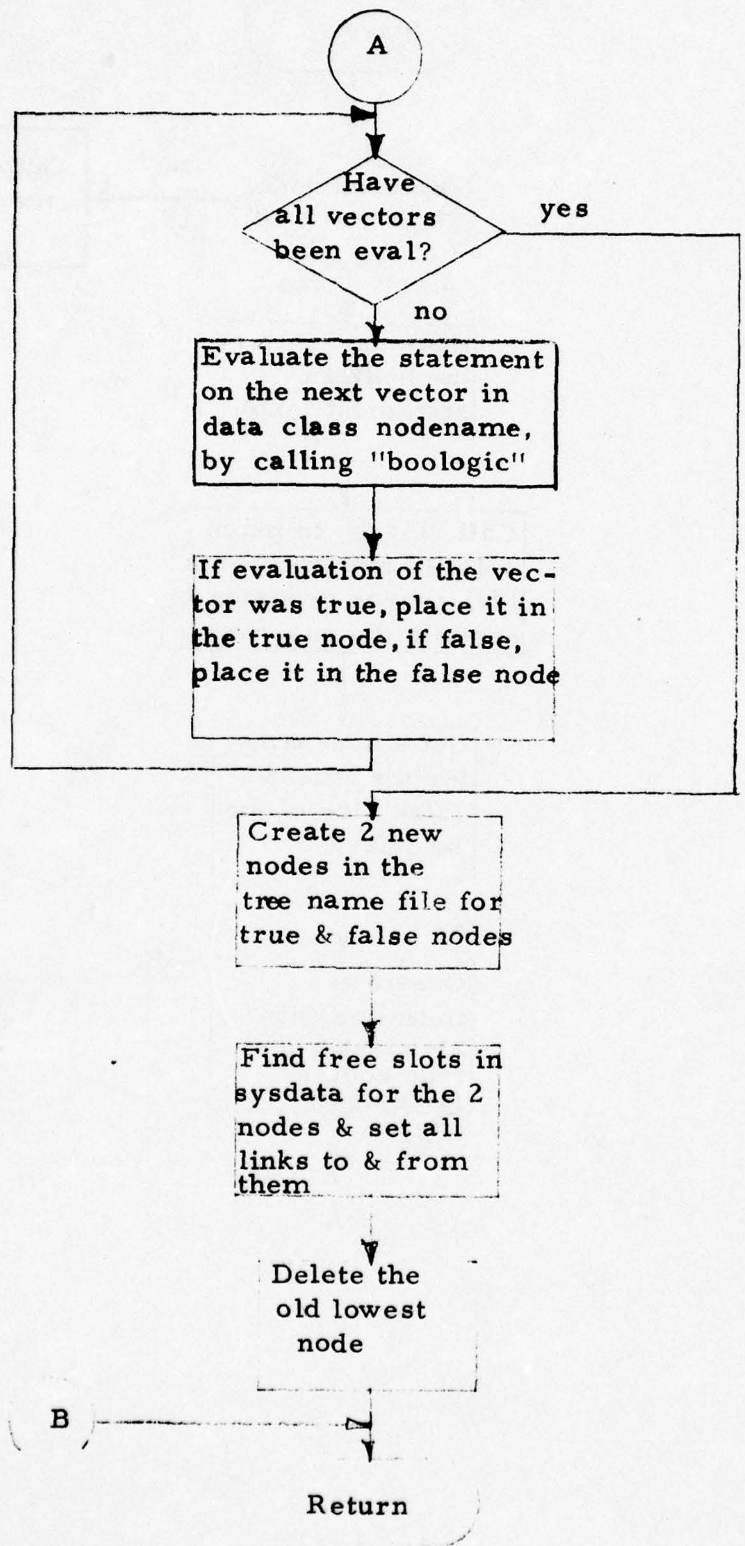
2 new classes are added to sysdata to reflect the tree structure after partitioning.

Program Description: "lingpart" is a structure analysis routine for dividing a class into 2 classes through the use of a boundary. This boundary takes the form of a user-entered boolean, linguistic statement. The program reads the entered statement, places it in a PL/1 source statement to do the partitioning, calls the PL/1 compiler to compile the new program, and then calls this newly compiled program to actually do the partitioning.

Flow Chart: Next page.

lingpart


Start
Is entered node a lowest node?
no
Output error message
B
yes
Have user enter the boolean statement (exp)
Call "logen" to place entered statement in a PL/1 source program & compile the program
Get a nodename for the true & false side of the boundary
Create & initialize data class files for true node & false node
A


A
Have all vectors been eval?
yes
no
Evaluate the statement on the next vector in data class nodename, by calling "boologic"
If evaluation of the vector was true, place it in the true node, if false, place it in the false node
Create 2 new nodes in the tree name file for true & false nodes
Find free slots in sysdata for the 2 nodes & set all links to & from them
Delete the old lowest node
B
Return

Internal Subroutine Name: lingpart$slot

Calling Sequence: call lingpart$slot (ptr, jinx)

Input Parameters:

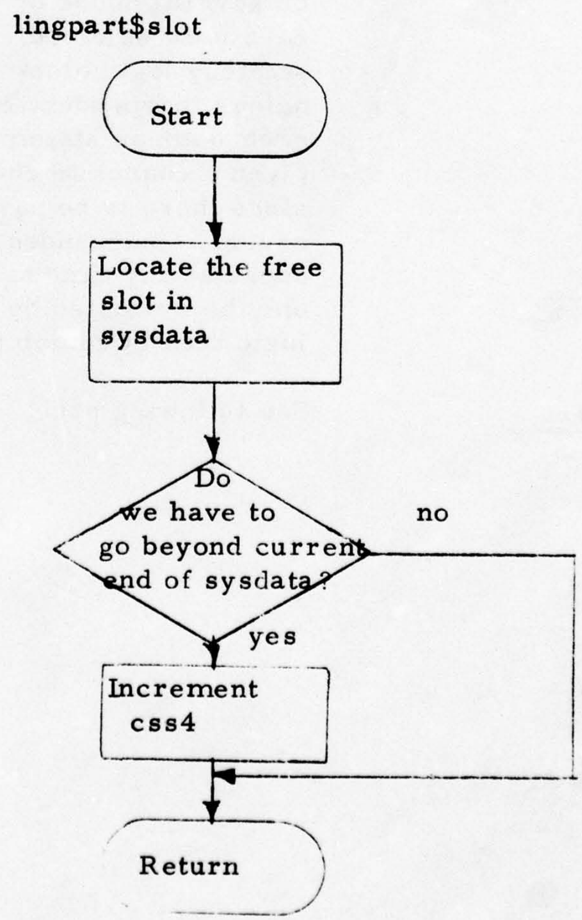
ptr - a pointer to the sysdata file

Output Parameters:

jinx - a relative index from the beginning of sysdata to the next free slot in the school entry of sysdata

Program Description: "lingpart$slot" returns the relative index (jinx) in sysdata to the next free slot in the school entry of sysdata. If this slot is beyond the last node (css4 of sysdata), css4 is incremented by 1.

Flow Chart:



MOOS Function Name: lingrjct

MOOS Function Number: 61

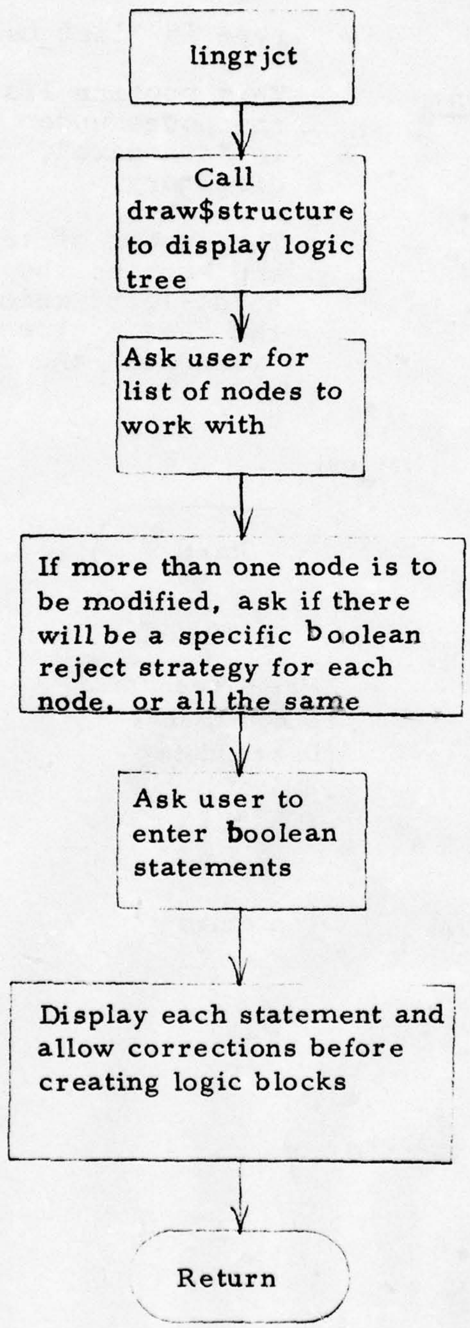
Calling Sequence: Type in "lingrjct (treename) (nodename) "

Input Parameters: Standard optional data set selection parameters

Output File Settings: lingrjct modifies node entries in the node part of the structure part of the mooslogic file to indicate that independent reject strategies exist and also creates independent reject strategy logic blocks.

Program Description: lingrjct calls draw$structure to display the selected logic tree and asks the user to input a list of logic nodes to which he wants to add independent reject strategies. The user can have one strategy created which will operate on several nodes or specific strategies for each node selected. An independent reject strategy logic block is created for each unique independent reject strategy. After each boolean statement is input, the user is given a chance to change the statement - since there is no partial evaluation to detect errors. Independent reject logic can be added at any time to any logic node, but can only be evaluated by "logicevl" when the logic tree is complete.

Flow Chart: See following page.


lingrjct
Call draw$structure to display logic tree
Ask user for list of nodes to work with
If more than one node is to be modified, ask if there will be a specific boolean reject strategy for each node, or all the same
Ask user to enter boolean statements
Display each statement and allow corrections before creating logic blocks
Return

Utility Function Name: list_cst

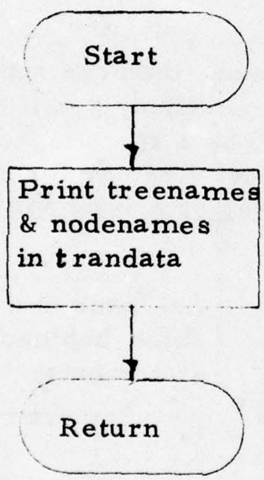
Calling Sequence: type in "list_cst"

Program Description: This routine lists the trees and the nodes under these trees present in "trandata", the common access directory.

The number of trees is determined and becomes the control index for a "do-loop" which processes through the "seg_o_ trees" segment in "trandata" and lists the data names.

Flow Chart:

list_cst



Utility Function Name: list_ust

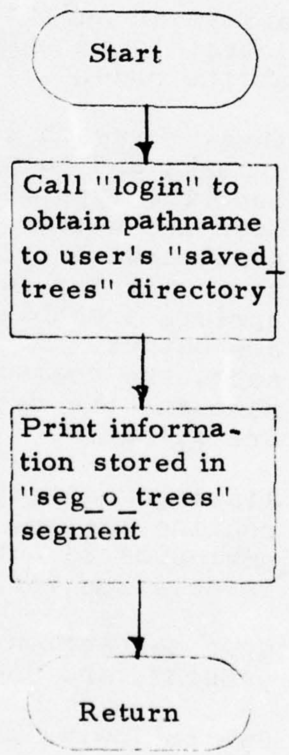
Calling Sequence: type in "list_ust"

Program Description: This routine lists the data trees and nodes under these trees that the user has placed in his "saved_trees" directory through the utility function "save".

The information to be printed is obtained from the "seg_o_trees" segment present in the user's directory.

Flow Chart:

list_ust



Utility Function Name: listlogc

Calling Sequence: type in "listlogc [(treename)] [(nodename)]"

Input Parameters: the standard optional data set selection parameters

Function Description: listlogc creates the output file "listlog_file") and unmasks and outputs the following information contained in the treename, nodename mooslogic file: the data set,the number of dimensions, and the number of lowest data class nodes.

Next, listlogc builds the parallel arrays ln and lc, which pairs each lowest logic node with the data class at the node.

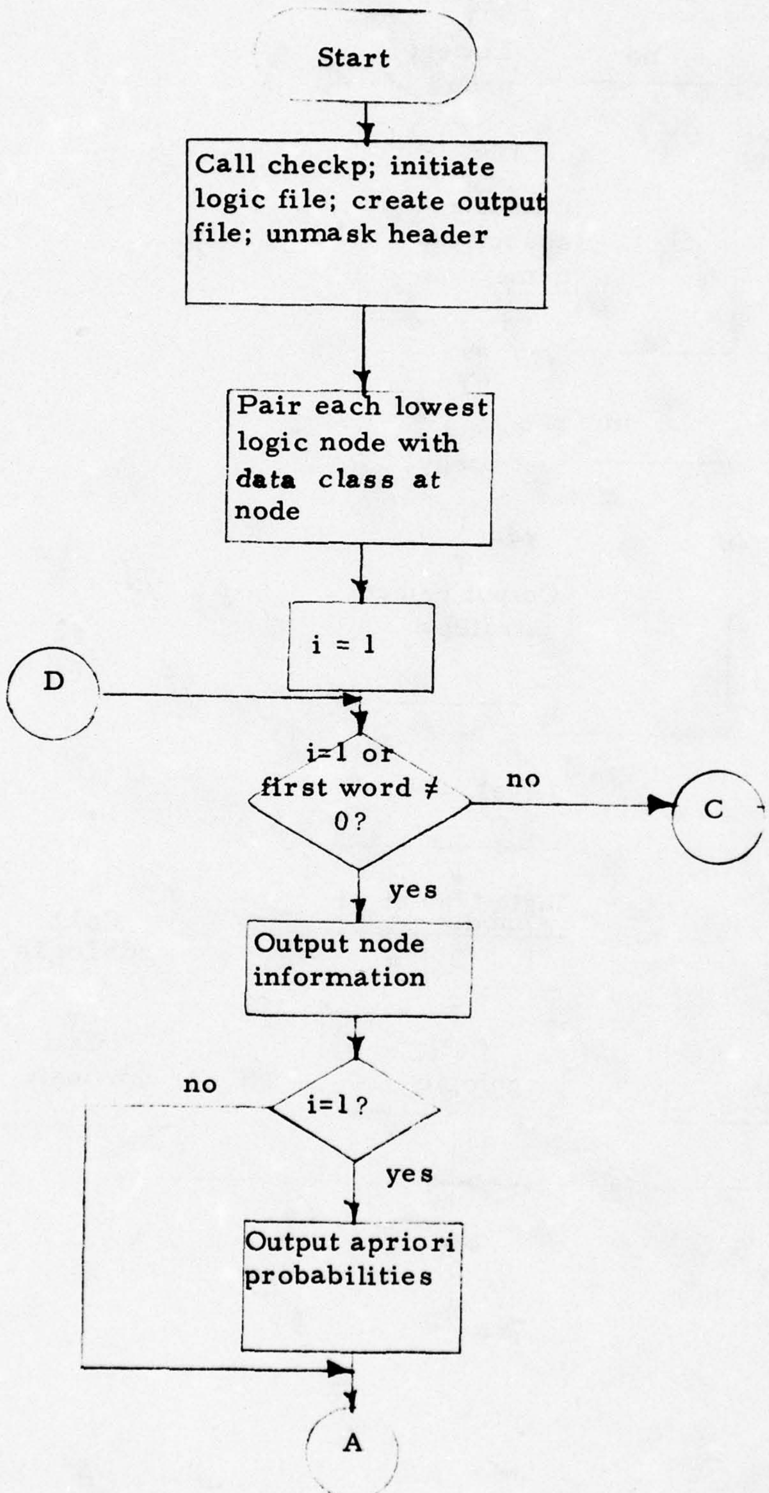
Then, for each logic node, the routine unmasks and outputs the logic node number, the type of logic, the superior node and number of nodes below the node, and the classes present at the node. If the logic node type is 1, apriori probabilities of each class are output. If the node is a lowest node, the reassociated data class name and the reject strategy (if any) are output.

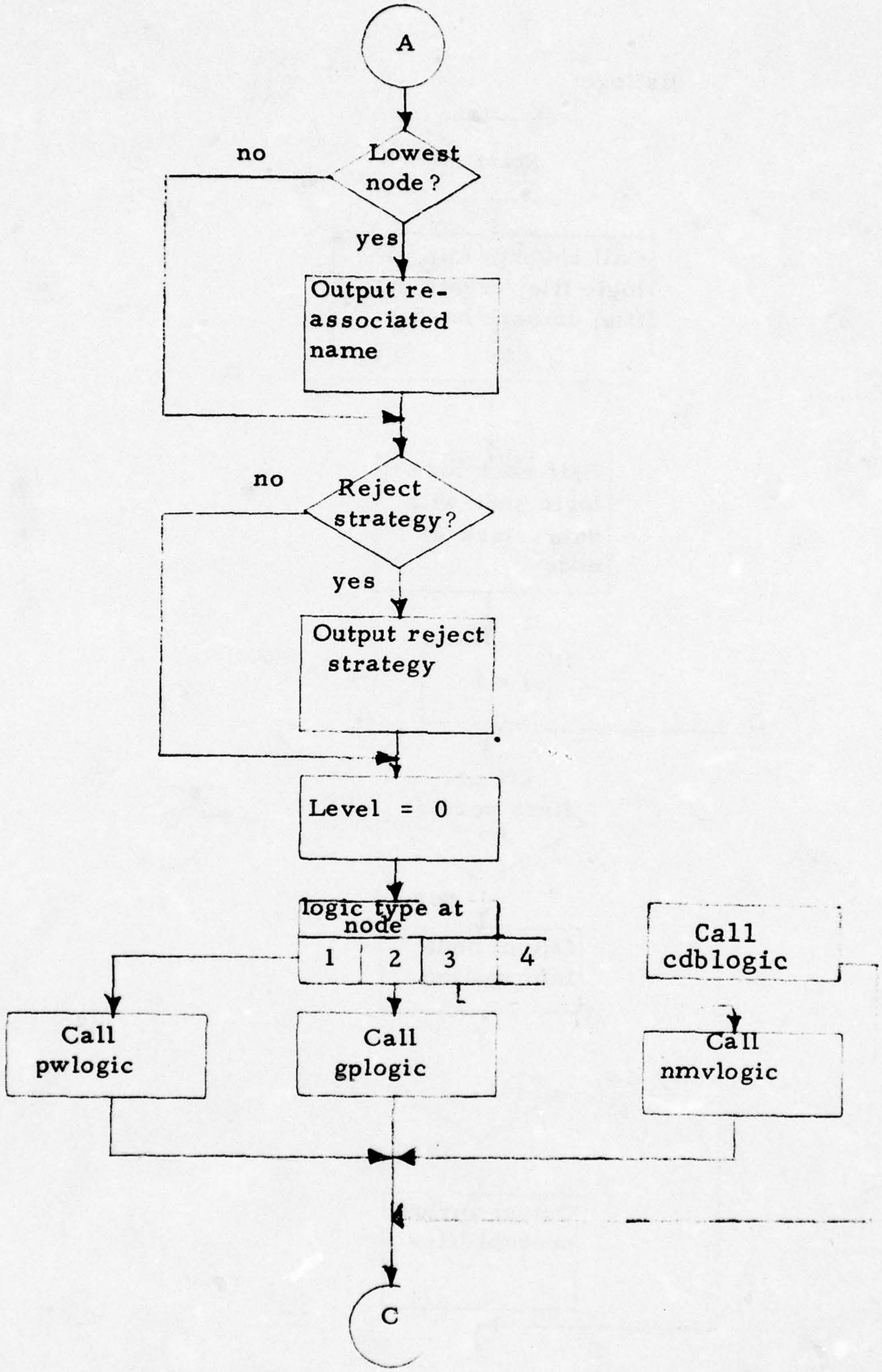
listlogc calls the appropriate subroutine (pwlogic , gplogic, cdblogic, or nmvlogic) to output the specific information for each node.

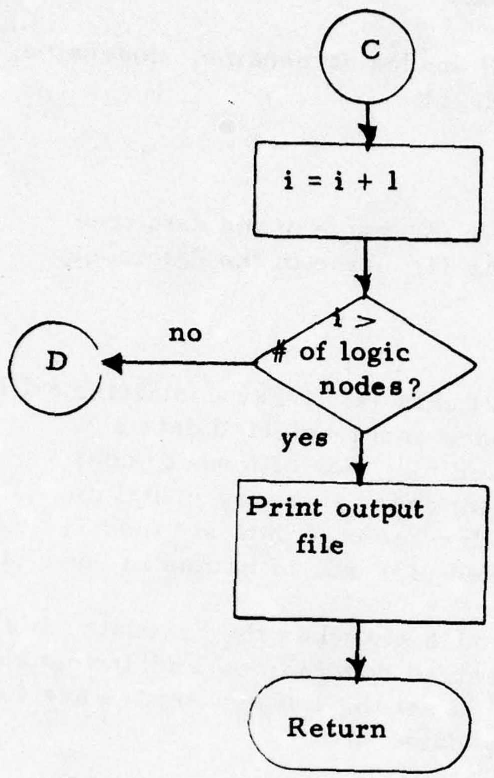
Upon completion, the output file is printed, and the routine returns.

Flow Chart: See following page

listlogc


Start
Call checkp; initiate logic file; create output file; unmask header
Pair each lowest logic node with data class at node
i = 1
D
i=1 or first word ≠ 0?
no
C
yes
Output node information
no
i=1?
yes
Output apriori probabilities
A


A
Lowest node?
no
yes
Output re-associated name
Reject strategy?
no
yes
Output reject strategy
Level = 0
logic type at node
1
2
3
4
Call pwlogic
Call gplogic
Call cdblogic
Call nmvlogic
C


C
i = i + 1
i >
# of logic
nodes?
no
D
yes
Print output
file
Return

Internal Subroutine Name: lnodes

Calling Sequence: call lnodes (treename, nodename, low, l, nod, n)

Input Parameters:

treename	-	char (8) name of the data tree.
nodename	-	char (4) name of the data node.

Output Parameters:

low	-	(72) char (4) array containing all lowest nodes under selected data set.
l	-	fixed (35) no. of lowest nodes.
nod	-	(142) char (4) array containing all nodes. under selected data set (nod(1) = nodename).
n	-	fixed (35) no. of names in "nod" array.

Program Description: lnodes searches the "sysdata" file for the desired node names and lowest node names. n is set to -1 if any errors are found in sysdata.

Flow Chart:

lnodes

↓

Search sysdata for lowest & intermediate nodes

↓

Return

Utility Function Name: log

Calling Sequence: Type in "log$save [(treename)] [(nodename)] "
Type in "log$rstr [(treename)] [(nodename)] "
Type in "log$dlet [(treename)] [(nodename)] "
Type in "log$list"

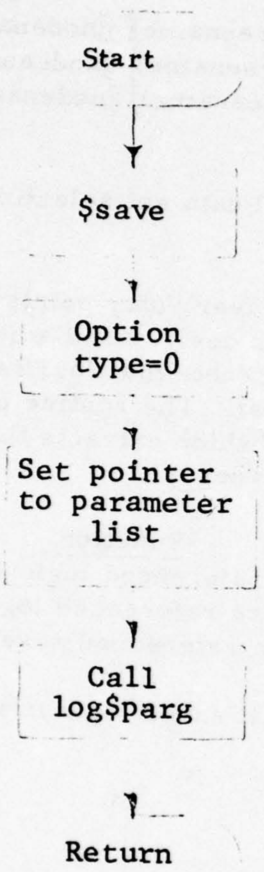
Input Parameters: The standard optional data set selection parameters

Program Description: Log consists of four user entry points which set the option number desired and a pointer to the parameter list (note that log$list requires no parameters). The routine passes control to log$parg, which extracts the parameters and calls log_p.

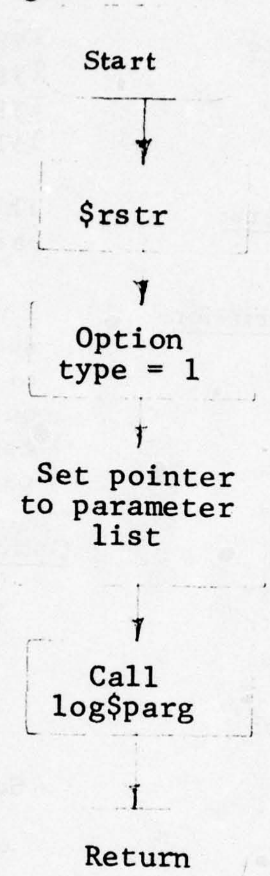
Entry	Option No.	Function
log$save	0	save referenced logic file
log$rstr	1	restore referenced logic file
log$dlet	2	delete referenced saved logic file
log$list	3	list all saved logic files

Flow Chart: See following page.

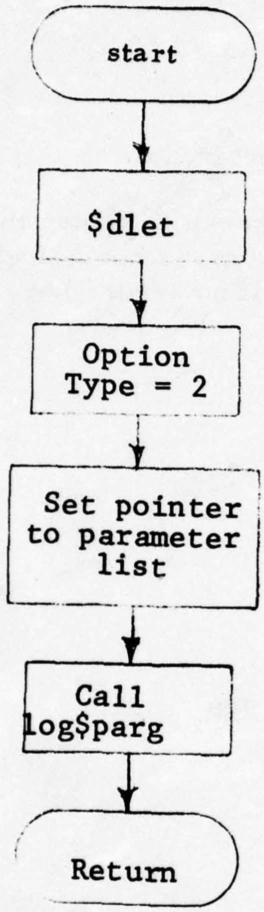
log$save



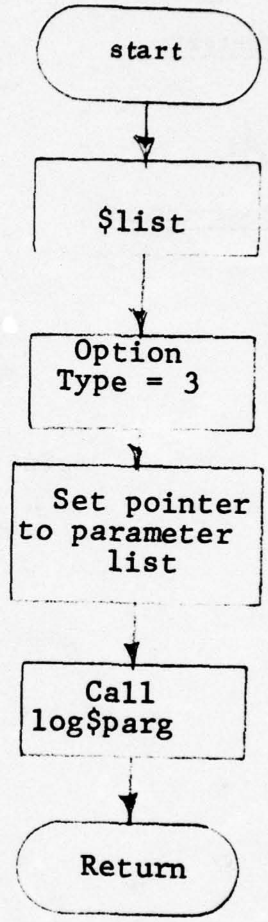
log$rstr



log$dlet



log$ list



Internal Subroutine: log$parg

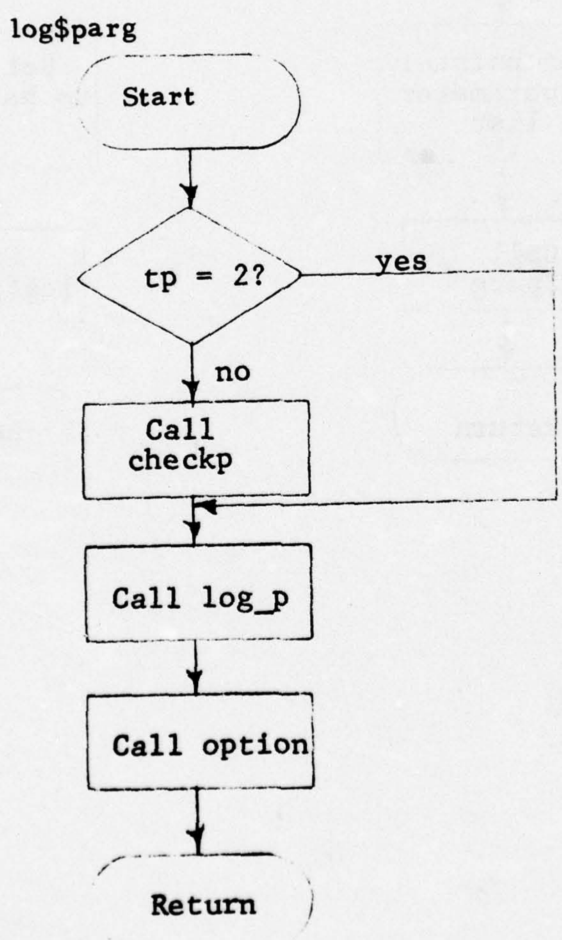
Calling Sequence: call log$parg (tp, parptr)

Input Parameters:

tp	-	option desired.
parptr	-	pointer to parameter list.

Program Description: log$parg calls checkp to return the parameter list if the option type is not 2 (log$list), and then calls log_p if no errors have occurred.

Flow Chart:



Internal Subroutine Name: log_p

Calling Sequence: call log_p (type, parml, parm2)

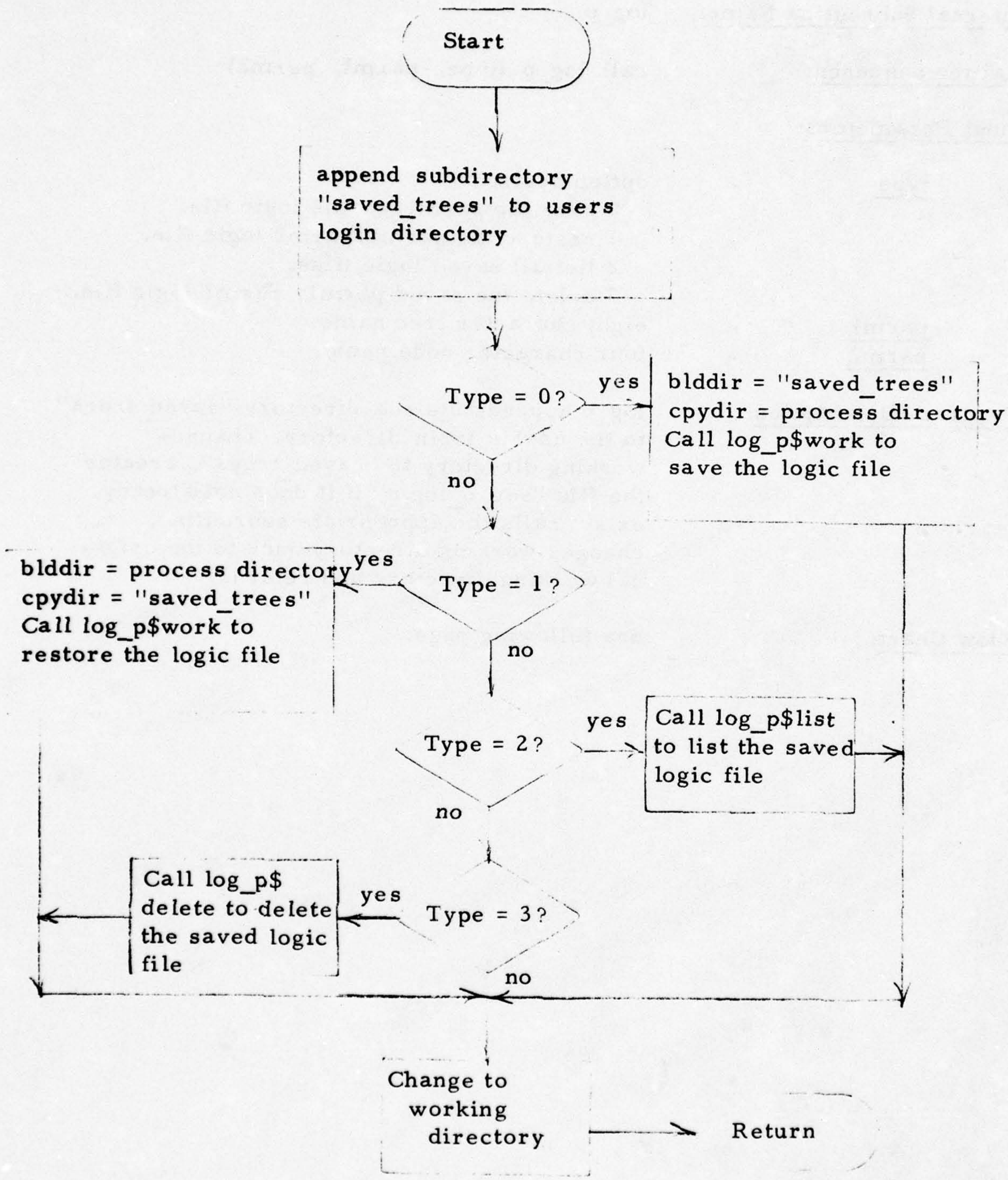
Input Parameters:

type	-	option desired 0 save the parml, parm2 logic file. 1 restore the parml, parm2 logic file. 2 list all saved logic files. 3 delete the saved parml, parm2 logic file.
parml	-	eight character tree name.
parm2	-	four character node name.

Program Description: log_p appends the sub-directory "saved_trees" to the user's login directory, changes working directory to "saved_trees", creates the file "seg_o_logic" if it does not already exist, calls the appropriate subroutine, changes working directory back to the original working directory and returns.

Flow Chart: See following page.

log_p


Start
append subdirectory "saved_trees" to users login directory
Type = 0?
yes
blddir = "saved_trees"
cpydir = process directory
Call log_p$work to save the logic file
no
Type = 1?
yes
blddir = process directory
cpydir = "saved_trees"
Call log_p$work to restore the logic file
no
Type = 2?
yes
Call log_p$list to list the saved logic file
no
Type = 3?
yes
Call log_p$ delete to delete the saved logic file
no
Change to working directory
Return

log_p: "seg_o_logic" file

1	SL1	SL1 - # of saved logic files
2	SL2(1)	SL2(i) - 8-character tree name reference for saved logic file(i)
3	SL2(1)	
4	SL3(1)	SL3(i) - 4-character node name reference for saved logic file(i)
5	SL2(2)	
6	SL2(2)	
7	SL3(2)	
	. . .	
(SL1-1)*3+2	SL2(SL1)	
(SL1-1)*3+3	SL2(SL1)	
(SL1-1)*3+4	SL2(SL1)	

Internal Subroutine Name: log_p$delete

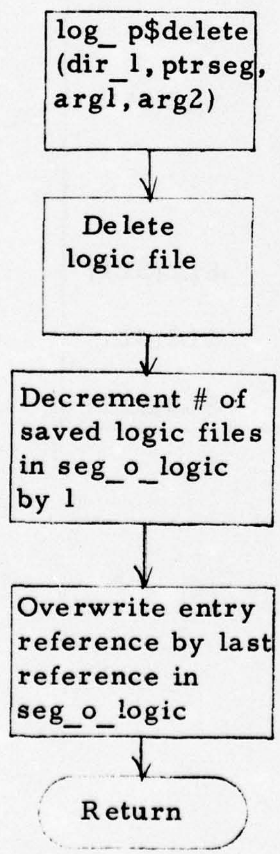
Calling Sequence: call log_p$delete (dir_1, ptrseg, arg1, arg2)

Input Parameters:

dir_1	-	user's login sub-directory "saved_trees".
ptrseg	-	pointer to "seg_o_logic" file.
arg1	-	eight character tree name.
arg2	-	four character node name.

Program Description: This routine deletes a previously saved logic file and deletes the reference from the seg_o_logic file. An error message is printed if no logic file under this reference is found in "saved_trees".

Flow Chart:



Internal Subroutine Name: log_p$list

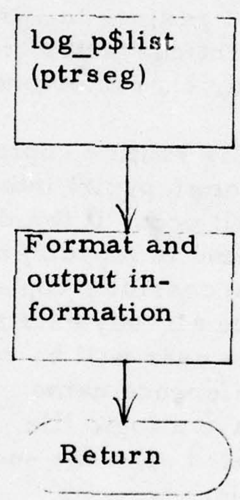
Calling Sequence: call log_p$list (ptrseg)

Input Parameter:

ptrseg - pointer to "seg_o_logic file.

Program Description: This routine formats and outputs to the screen the number and references of saved logic files as listed in "seg_o_logic".

Flow Chart:



Internal Subroutine Name: log_p$work

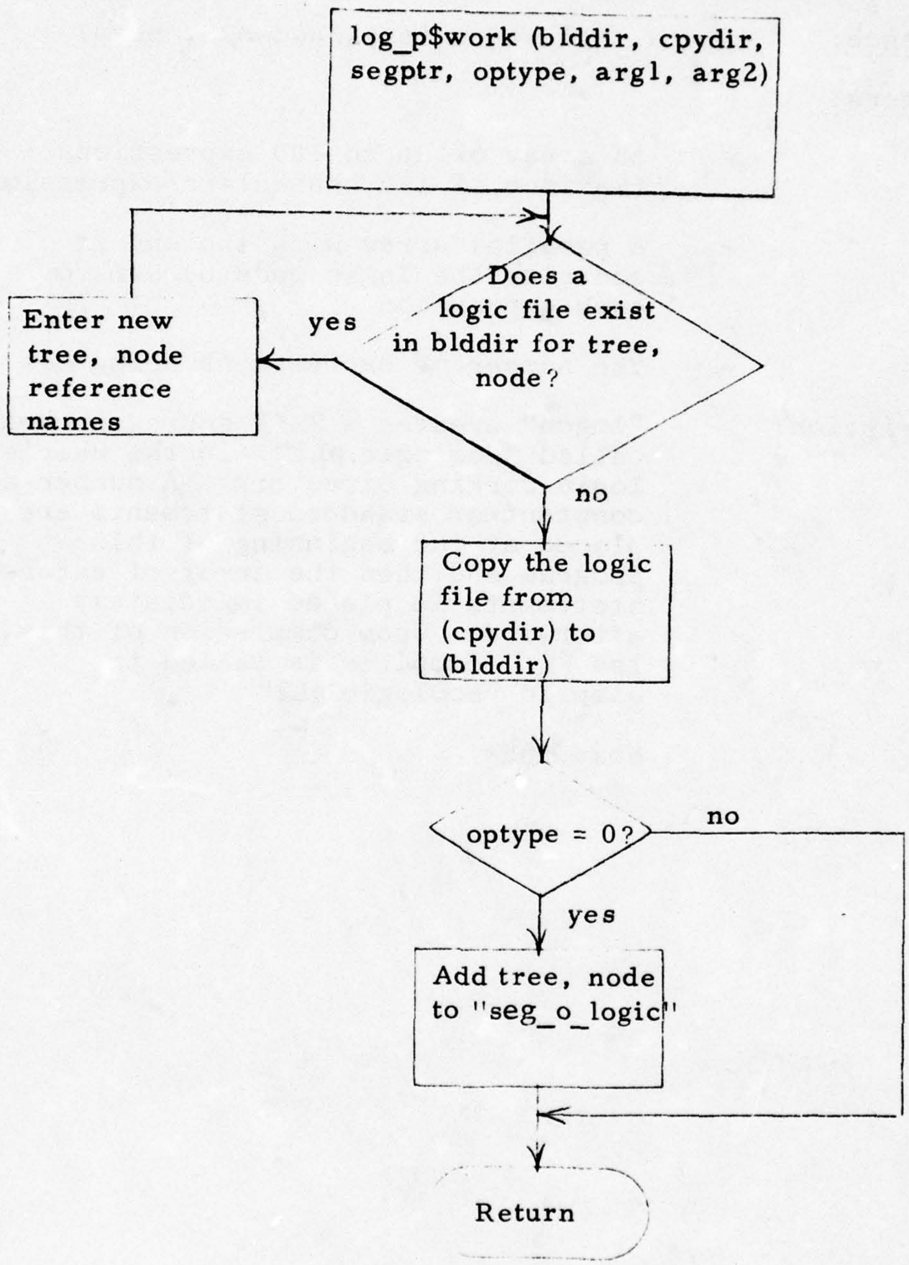
Calling Sequence: call log_p$work (blddir, cypdir, segptr, optype, argl, arg2)

Input Parameters:

- blddir - directory in which the argl, arg2 logic file is to be copied.
- cpydir - directory from which the argl, arg2 logic file is to be copied.
- segptr - pointer to the "seg_o_logic" file.
- optype - option desired:
  - 0 save the argl, arg2 logic file.
  - 1 restore the argl, arg2 logic file.
- argl - eight character tree name.
- arg2 - four character node name.

Program Description: This routine copies the requested logic file from (cpydir) into (blddir). Error messages will occur if the desired logic file is not found in (cpydir) or if the segment is unsuccessfully copied into (blddir). If a logic file already exists under the given name, the user will be requested to supply new reference name. If the option desired is to save a logic file, then "seg_o_logic" is updated upon the successful copying of the logic file.

Flow Chart: See following page.


log_p$work (blddir, cpydir, segptr, optype, arg1, arg2)
Does a logic file exist in blddir for tree, node?
yes
Enter new tree, node reference names
no
Copy the logic file from (cpydir) to (blddir)
optype = 0?
no
yes
Add tree, node to "seg_o_logic"
Return

Internal Subroutine Name: logen

Calling Sequence: call logen (exp, nodenum, nexp)

Input Parameters:

exp - an array of up to 100 expressions (maximum of 132 characters/expression)

nodenum - a parallel array with exp and it contains the logic node number for each expression

nexp - the number of expressions being set

Program Description: "logen" creates a PL/1 source program called "boologic.pL1" in the user's login working directory. A number of constant or standard statements are placed at the beginning of this program and then the array of entered statements is placed immediately afterward. Upon completion of this, the PL/1 compiler is called to compile "boologic.pL1"

Flow Chart: Next page

logen

Start

Create segment "boologic. p Ll" in user's login directory

Place the statements:

```
"boologic: pnc(ptr, ndim, boolval, num);
 dcl(ptr, aptr) ptr;
 dcl(i, ndim, num) fixed bin (35);
 dcl  boolval bit (1);
 dcl  m(100) float;
 dcl  flot float based;
        aptr = ptr
     do  i = 1 to ndim;
          m(i) = (aptr-> flot);
          aptr = addrel (aptr,  1);
     end;"
```

in "boologic. p L1".

Insert all "exp" expressions into "boologic. p Ll" next

Place the statements: "return; end;" into "boologic. p Ll" next.

A

A

Compile "boologic.pLl" by calling PL/l compiler to create segment "boologic"

Delete segment "boologic.pLl"

Return

MOOS Program Name: logicevl

MOOS Program Number: 64

Calling Sequence: Type"logicevl [(treename)] [(nodename)]"

Input Parameters: Standard optional data set selection parameters

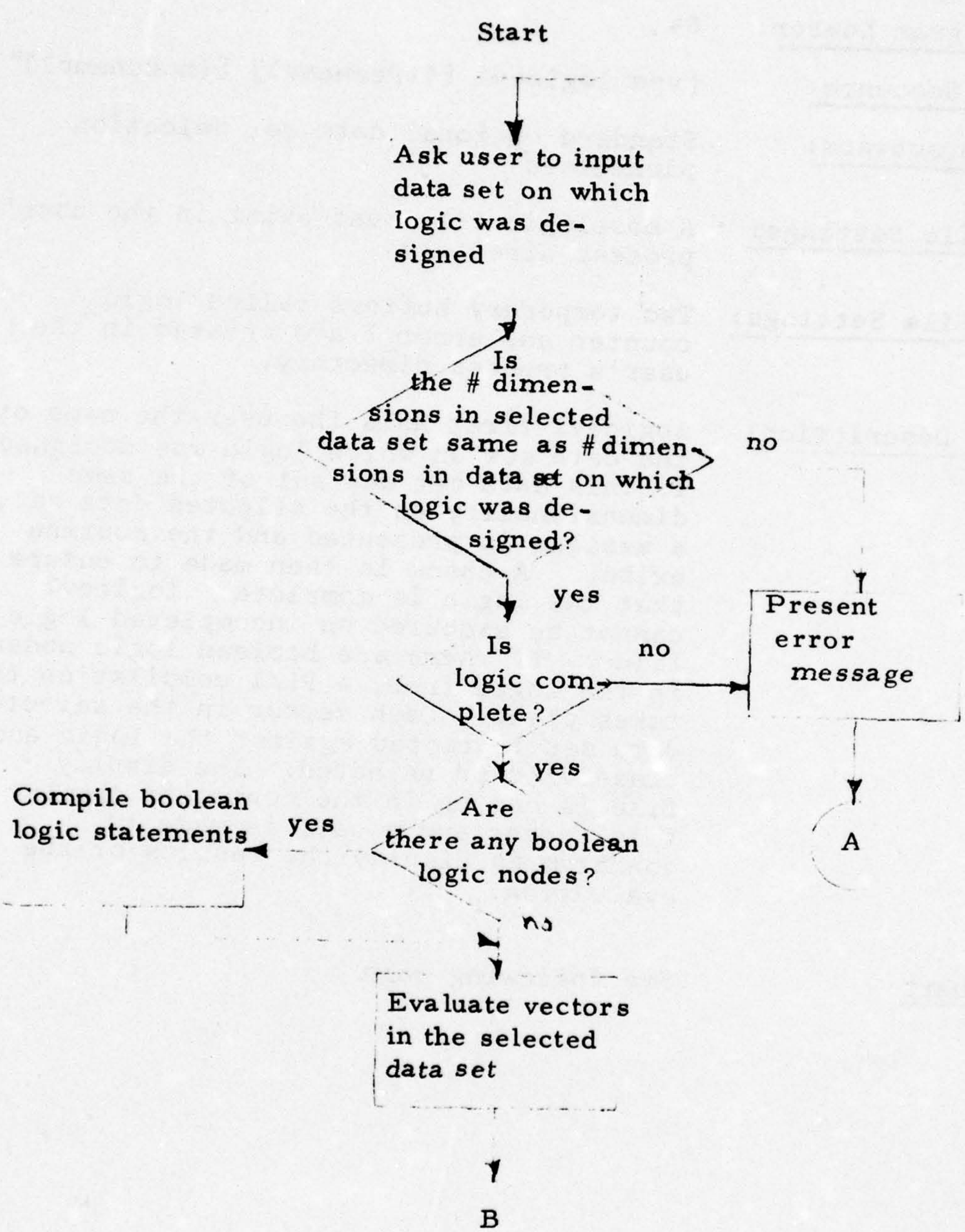
Input File Settings: A mooslogic file must exist in the user's process directory

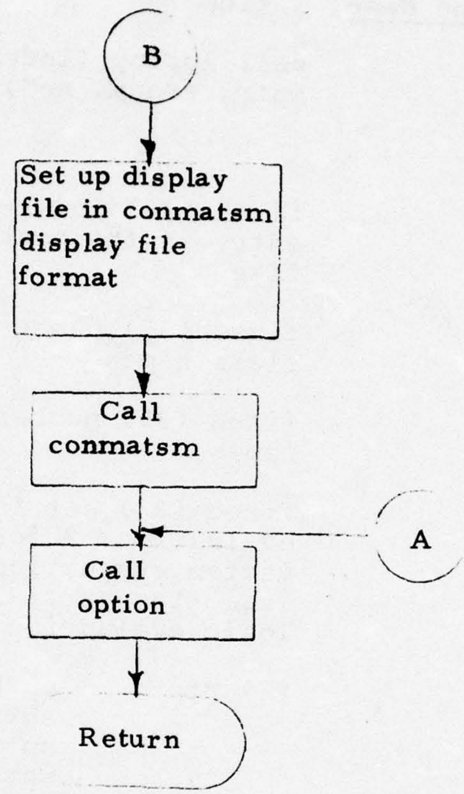
Output File Settings: Two temporary buffers called logic_counter and error_f are created in the user's process directory.

Program Description: logicevl first asks the user the name of the data set on which logic was designed. If this data set was not of the same dimensionality as the selected data set, a message is presented and the routine exits. A check is then made to ensure that the logic is complete. logicevl cannot be executed on incompleted logic files. If there are boolean logic nodes in the logic tree, a PL/1 compilation then takes place. Each vector in the selected data set is tested against the logic and classified or rejected. The display file is set up in the conmatsm display file format and a call is made to conmatsm to display the results of the evaluation.

Flow Chart: See following page

logicevl


Start
Ask user to input data set on which logic was designed
Is the # dimensions in selected data set same as # dimensions in data set on which logic was designed?
no
yes
Present error message
A
Is logic complete?
no
yes
Are there any boolean logic nodes?
yes
Compile boolean logic statements
no
Evaluate vectors in the selected data set
B


B
Set up display file in conmatsm display file format
Call conmatsm
Call option
A
Return

Internal Subroutine Name: logicp

Calling Sequence: call logicp (index, lncls, tncls, ii, sptr, trnam, nod)

Input Parameters:

index — fixed (35) index to the first C28 entry in the confusion matrix display file

lncls — fixed (35) number of "assigned" class names

tncls — fixed (35) number of "true" class names

ii — fixed (35) set to 1 for overall logic evaluation, 3 for partial nearest mean vector evaluation; any other value indicates partial pairwise or group logic evaluation

sptr — (5) ptr
sptr(1) - sysdata
sptr(2) - scratch
sptr(3) - display
sptr(4) - treename
sptr(5) - mooslogic

trnam — char(8) tree name of the data set being evaluated

nod — char(4) node name of the data set being evaluated

Input File Settings: The display file must be set up according to the confusion matrix display file format. In the case of partial pairwise evaluation, some information must also be stored in the scratch file.

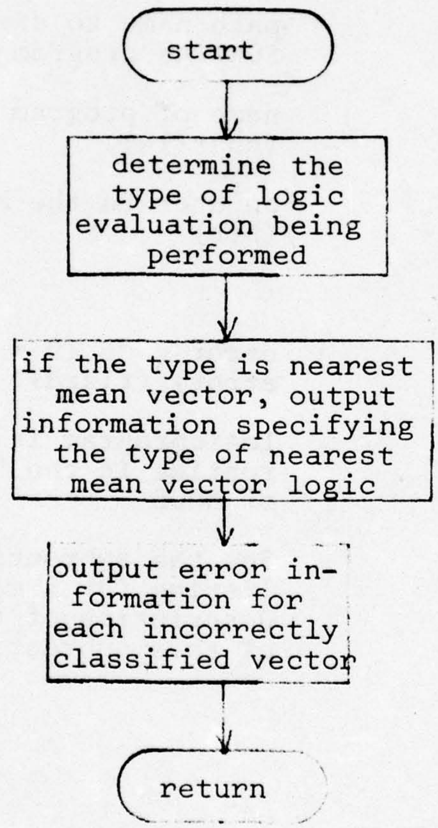
A temporary file called error_f must also exist as well as certain specific temporary error files. These error files are described in detail in section 3.

Program Description: Logicp prints the detailed error listing which the user may select after overall evaluation, partial pairwise

evaluation, or partial nearest mean vector evaluation. It retrieves the necessary information from the error_f, pair_error_file, nmv_error_file, box_error_file, display, scratch, and mooslogic files.

Flow Chart:

logicp



Internal Subroutine Name: logicprogram

Calling Sequence: call logicprogram (path, file, logicptr, error)

Input Parameters:

path	path name to directory to contain FORTRAN program (char(168))
file	name of program to be created (char(14))
logicptr	pointer to the MOOS logic file (ptr)

Output Parameters:

error	error code (0 = no errors, 1 = error) (fixed)

Program Description: logicprogram is the executive routine in the "fortlogc" option of MOOS

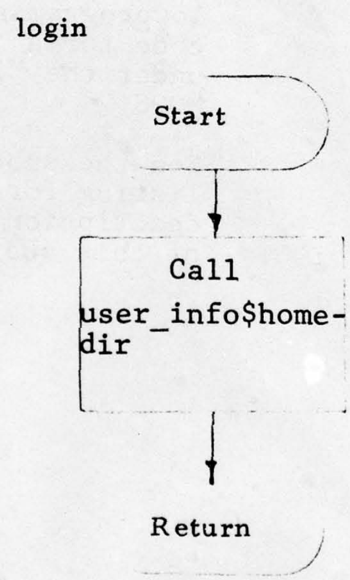
See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: login

Calling Sequence: call login (home_dir)

Program Description: The user's home directory is returned in the parameter "home_dir".

Flow Chart:



Internal Subroutine Name: lowprogram

Calling Sequence: call lowprogram (i, logicptr)

Input Parameters:

i — lowest logic node number (fixed (35))

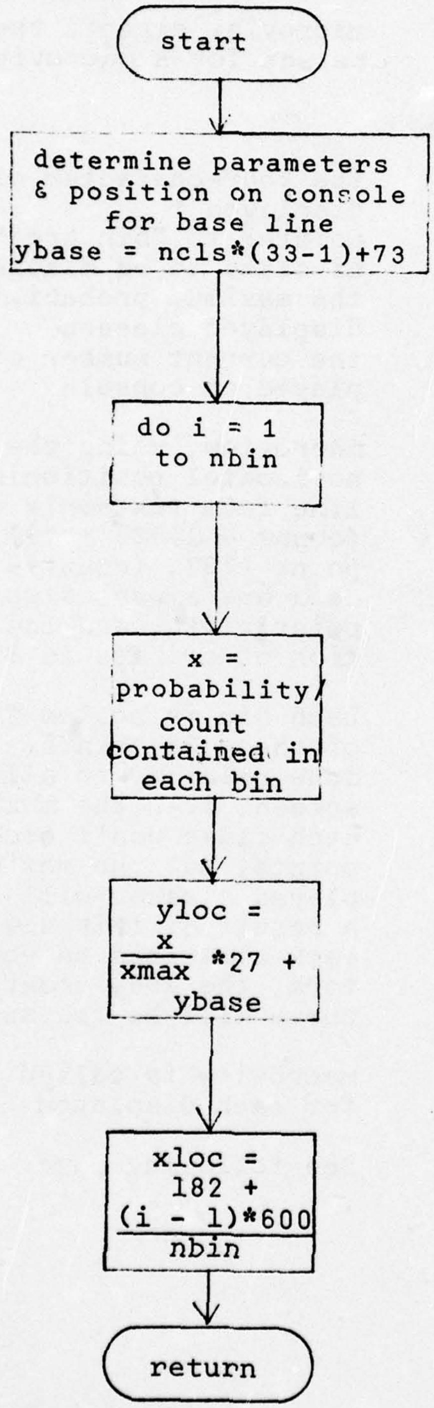
logicptr — pointer to the MOOS logic file (ptr)

Program Description: lowprogram generates FORTRAN code for a lowest logic node under the "fortlogc" option of MOOS

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name:	macroview
Calling Sequence:	call macroview (name, ptr, max, count)
Input File Settings:	macroview expects the "csdata" file to be set for a macroview display.
Input Parameter Settings:	
name	the four-character class name to be displayed
ptr	pointer to "bin area" data of "csdata" of class to be displayed
max	the maximum probability/count of all displayed classes
count	the current number of classes displayed on console
Program Description:	macroview, using the number count for horizontal positioning, draws a base line from tektronix point (182, (count - 1)*33 + 73) to tektronix point (782, (count - 1)*33 + 73). The data are always displayed over a 600-point width, and the vertical separation of classes is 33 points. Each bin is scaled to a maximum of 27 of these 33 points. This scaling is done relative to all classes on the screen, i.e. the maximum bin height of each class won't necessarily be 27 points, but the maximum of all the displayed classes will be 27 points. As a result of this scaling procedure, if each class has an equal number of vectors, the area under each histogram curve will be the same. macroview is called by npcos or pcos for each displayed class.
Flow Chart:	See following page.

macroview


start
determine parameters & position on console for base line
ybase = ncls*(33-1)+73
do i = 1 to nbin
x = probability/count contained in each bin
yloc = x/xmax *27 + ybase
xloc = 182 + (i - 1)*600/nbin
return

Moos Function Name: measxfrm

Moos Function Number: 130

Calling Sequence: Type in "measxfrm [(treename)] [(nodename)]"

Input Parameters: Standard optional data set selection parameters

Output File Settings: A new tree is created in sysdata

A new tree name file is created

A new set of data class files is created

Program Description: "measxfrm" is a means of transforming a given data set into a new data set. This transformation takes the form of linguistic statements which are a function of the measurements from the original data set.
e.g. nm(1) = om(1) + om(2) says that measurement 1 of the new data set equals the sum of measurements 1 and 2 of the old data set.

"measxfrm" takes the entered statements, places them into a PL/1source program, compiles this PL/1source program, and then executes the newly compiled routine to do the transformation.

Flow Chart: Next page

measxfrm

Start

The new tree is a tree of lowest nodes, therefore get list of lowest nodes for old tree

Get new tree name

Get dimensionality of new tree

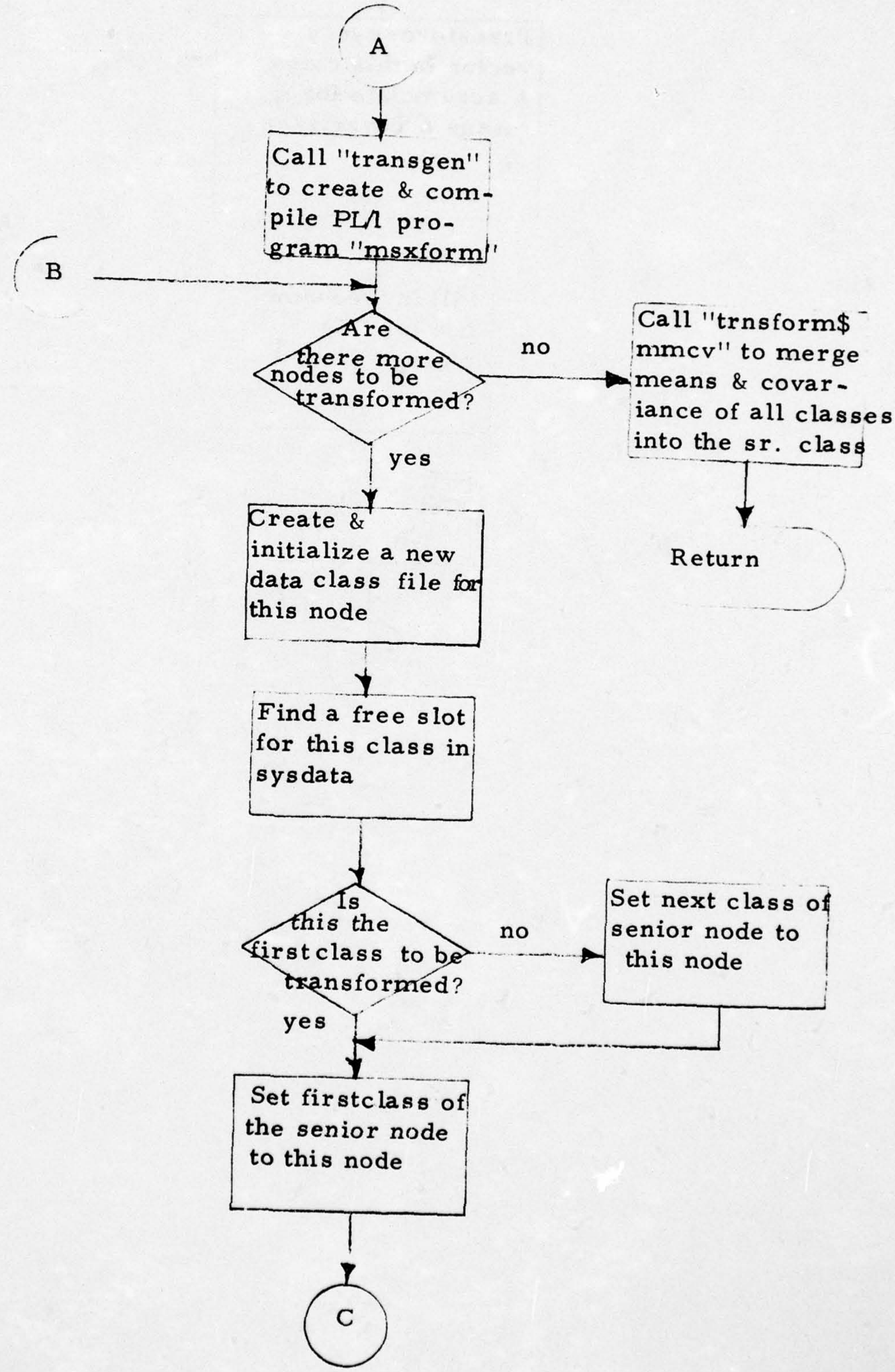
Find a free slot in sysdata for the new tree

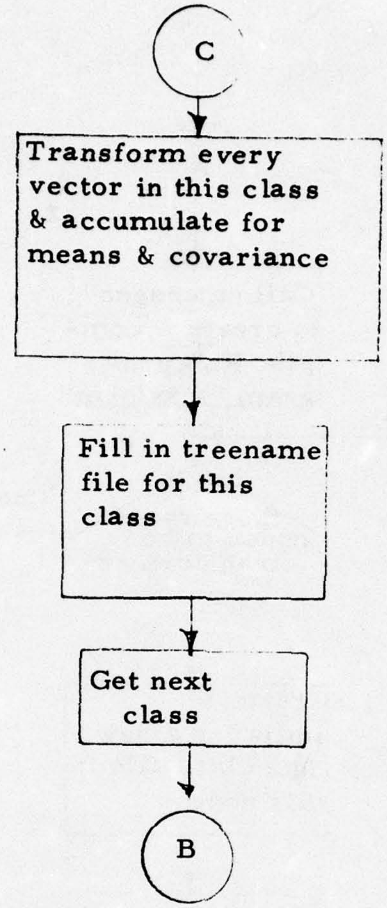
Fill in senior node of new tree

Create & initialize new treename file

Read transformation expressions (max of 75 expressions)

A


A
Call "transgen" to create & compile PL/1 program "msxform"
B
Are there more nodes to be transformed?
no
Call "trnsform$mmcv" to merge means & covariance of all classes into the sr. class
Return
yes
Create & initialize a new data class file for this node
Find a free slot for this class in sysdata
Is this the first class to be transformed?
no
Set next class of senior node to this node
yes
Set firstclass of the senior node to this node
C


C
Transform every vector in this class & accumulate for means & covariance
Fill in treename file for this class
Get next class
B

MOOS Function Name: mergmeas

MOOS Function Number: 4

Calling Sequence: Type in "mergmeas (newtree)"

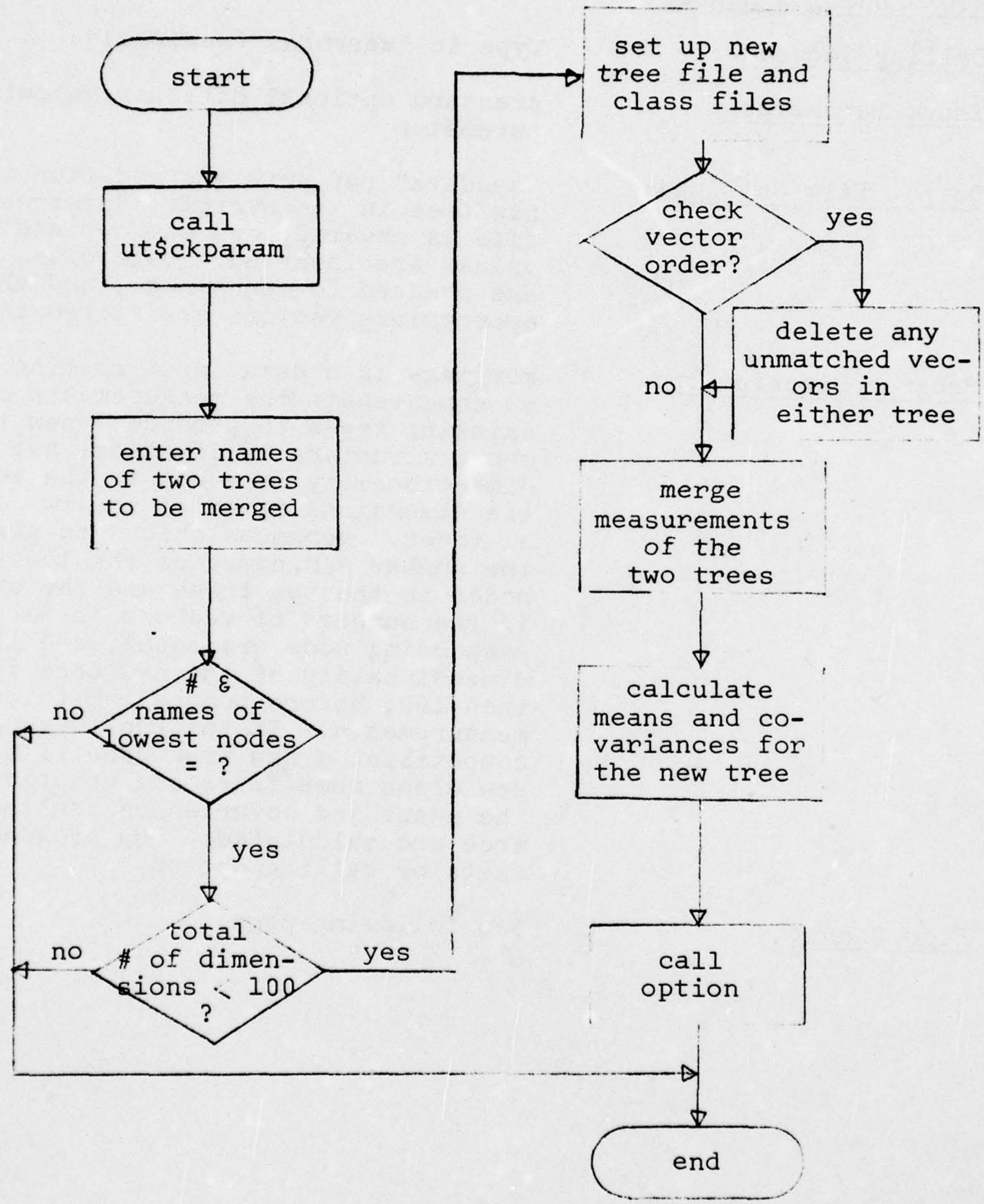
Input Parameters: standard optional data set selection parameter

Output File Settings: "sysdata" reflects the addition of a new tree in the system. "newtree" file is created, and appropriate values are inserted. Data class files are created for each node, and the appropriate vectors are stored in each.

Program Description: mergmeas is a data input routine used to concatenate the measurements of two existing trees to produce a new tree whose structure is identical but whose dimensionality is equal to the sum of the dimensionalities of the two original trees. mergmeas checks to see if the number and names of the lowest nodes in the two trees are the same, if the numbers of vectors in each corresponding node are equal, and if the dimensionality of the new tree is less than 100, before proceeding to combine measurements. If the two trees are compatible, a new tree name file and new class name files are created, and the means and covariances for the new tree are calculated. The program exits by calling option.

Flow Chart: See following page.

mergmeas


start
call ut$ckparam
enter names of two trees to be merged
& names of lowest nodes = ?
no
yes
total # of dimensions < 100 ?
no
yes
set up new tree file and class files
check vector order?
yes
delete any unmatched vecors in either tree
no
merge measurements of the two trees
calculate means and covariances for the new tree
call option
end

Internal Subroutine Name: microview

Calling Sequence: call microview (type)

Input File Setting: microview expects the "csdata" file and the "microbuff" file to be set for "micro" display

Input Parameter Setting:

type — type determines labelling
0 indicates probabilities
1 indicates counts

Program Description: microview is called once and produces the one-space "micro" display. Initially, microview determines the appropriate horizontal spacing. If the number of bins (nbin) is less than 15, then the display is 3 x nbin print positions otherwise its width is 600 tektronix points [wide, centered about the middle of the screen]. The maximum height is 475 "tek." points, relative to the base of the histogram, which is 73 "tek." points from the bottom of the display itself. The y-axis is labelled with either probabilities or counts, depending on the value of the parameter.

The horizontal positioning is determined as follows:

$$\text{xpos} = 182 + \text{constant}*12 + \frac{(j-1)*B}{\text{nbin}} \quad \text{where}$$

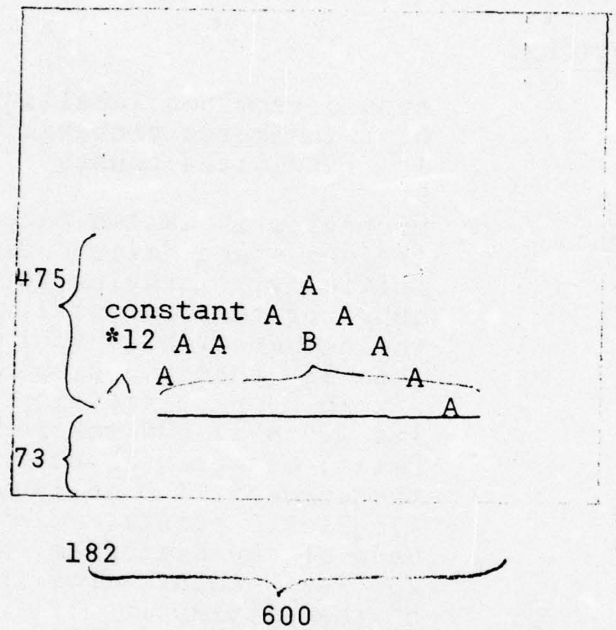
182 = offset, in "tek." points, from left boundary of the display screen

constant = offset, in print positions, from start of histogram, relative to 182. "constant" is a function of nbin where constant = 0 if nbin $\geq$ 15, else constant = $\frac{50-(3*\text{nbin})}{2}$

j = a do-loop index from do j=1 to nbin

nbin = number of bins

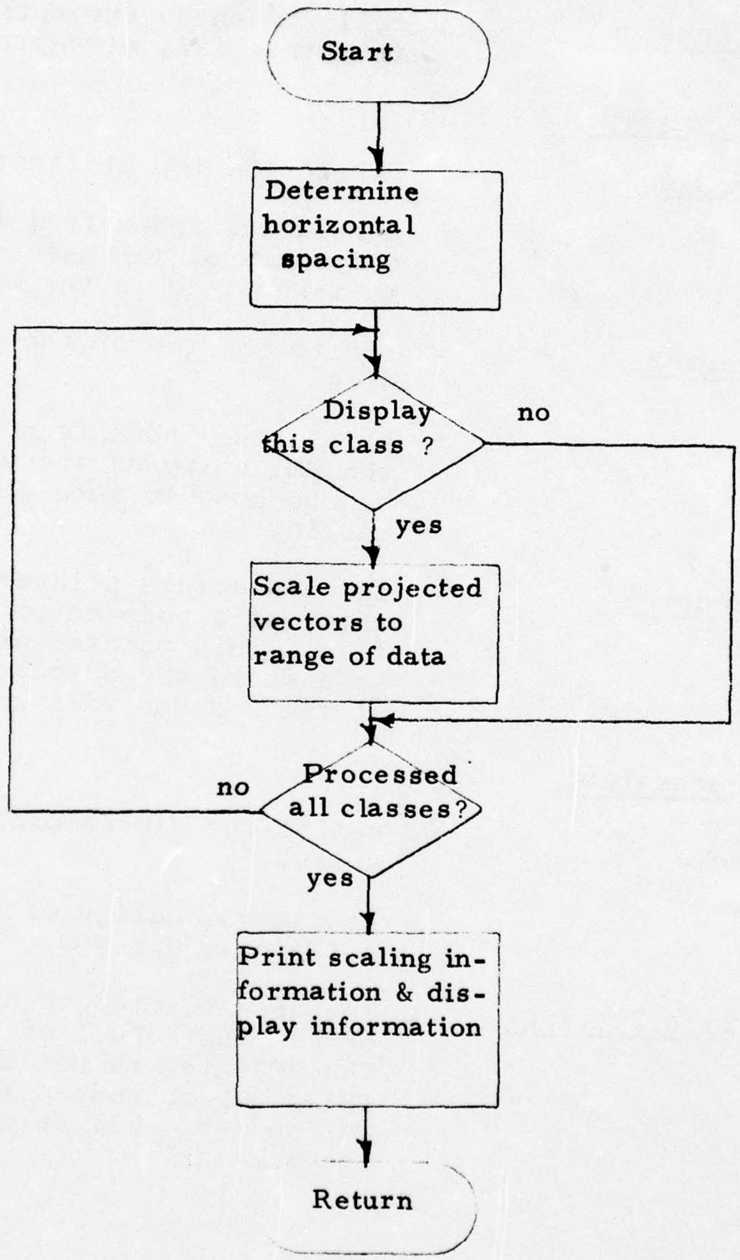
B = width of histogram in "tek." points
    B = 600 if constant = 0, else B = nbin*36



The y-axis information is stored in word D16 of the display file and is used in determining the scaling data. The routine exits by returning to the calling program.

Flow Chart: See following page

microview



Start
Determine horizontal spacing
Display this class ?
no
yes
Scale projected vectors to range of data
Processed all classes?
no
yes
Print scaling information & display information
Return

Internal Subroutine Name: mmeanacv

Calling Sequence: call mmeanacv (treeptr1, ix1, treeptr2, ix2, otreeptr, dim)

Input Parameters:

treeptr1 = ptr to the top of first tree name file

ix1 = a relative index from treeptr1 to the CM1 entry of the associated first node to take place in the merging

treeptr2 = ptr to the top of the second tree name file

ix2 = a relative index from treeptr2 to the CM1 entry of the associated second node to take place in the merging

otreeptr = is an absolute pointer to the CM1 entry of a node entry in a treename file. This pointer will point to the area where the output of the merging of the 2 nodes will go.

Output Parameter:

dim - set to the dimensionality of the 2 nodes.

If dimensionality of the 2 nodes are not equal, dim = 0

Output File Setting: The area pointed to by otreeptr will obtain the result of the merging. The CM entry is not touched. CM2 = CM2 of node 1 + CM2 of node 2 CM2 and CM4 will be changed to reflect the merging.

Program Description: mmeanacv merges the means and covariance matrices for 2 nodes; computation is done as follows:

Let A,B be the two nodes to combine

μ_{A_i}, μ_{B_i} = the i^{th} entry of the means for nodes A and B

μ_{AB_i} = the i^{th} entry of the mean for the combination of nodes A and B

N_A, N_B, N_{AB} = the number of vectors for node A, node B, node (A+B)

$\sigma_{A_{ij}}, \sigma_{B_{ij}}, \sigma_{AB_{ij}}$ = the (i,j) entry of the covariance matrix for node A, node B, node (A+B)

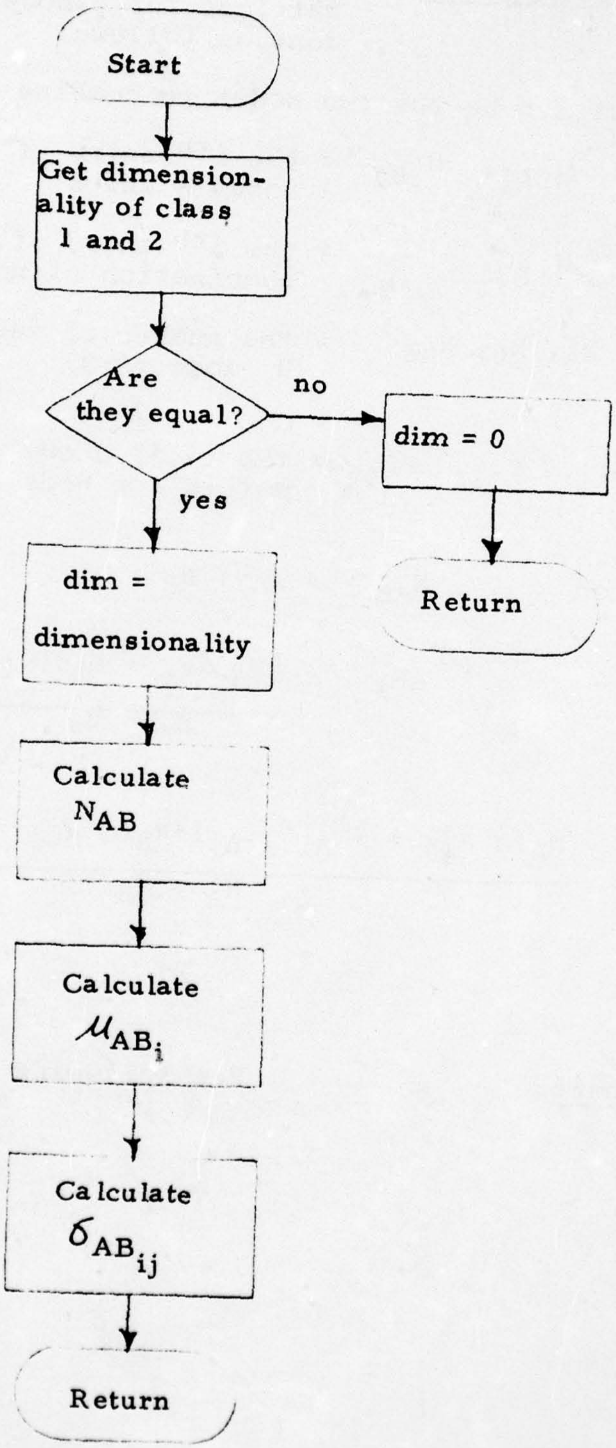
Then $N_{AB} = N_A + N_B$

$$\mu_{AB_i} = \frac{[N_A \mu_{A_i} + N_B \mu_{B_i}]}{N_A + N_B}$$

$$\sigma_{AB_{ij}} = \frac{N_A(\sigma_{A_{ij}} + \mu_{Ai} * \mu_{Aj}) + N_B(\sigma_{Bij} + \mu_{Bi} * \mu_{Bj})}{N_A + N_B} - \mu_{ABi} * \mu_{ABj}$$

Flow Chart: See following page

mmeanacv


Start
Get dimensionality of class 1 and 2
Are they equal?
no
dim = 0
Return
yes
dim = dimensionality
Calculate N_{AB}
Calculate μ_{AB_i}
Calculate $\sigma_{AB_{ij}}$
Return

AD-A033 437 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
MULTICS OLPARS OPERATING SYSTEM.(U)
SEP 76 D B CONNELL, K N KLINGBAIL F30602-75-C-0226
UNCLASSIFIED PAR-74-25-B RADC-TR-76-271-VOL-2 NL

5 OF 7
AD
A033437

Internal Subroutine Name: mncvotr

Calling Sequence: call mncvotr(ptrd, temp, ptro)

Input Parameters:

ptrd - pointer to data class file

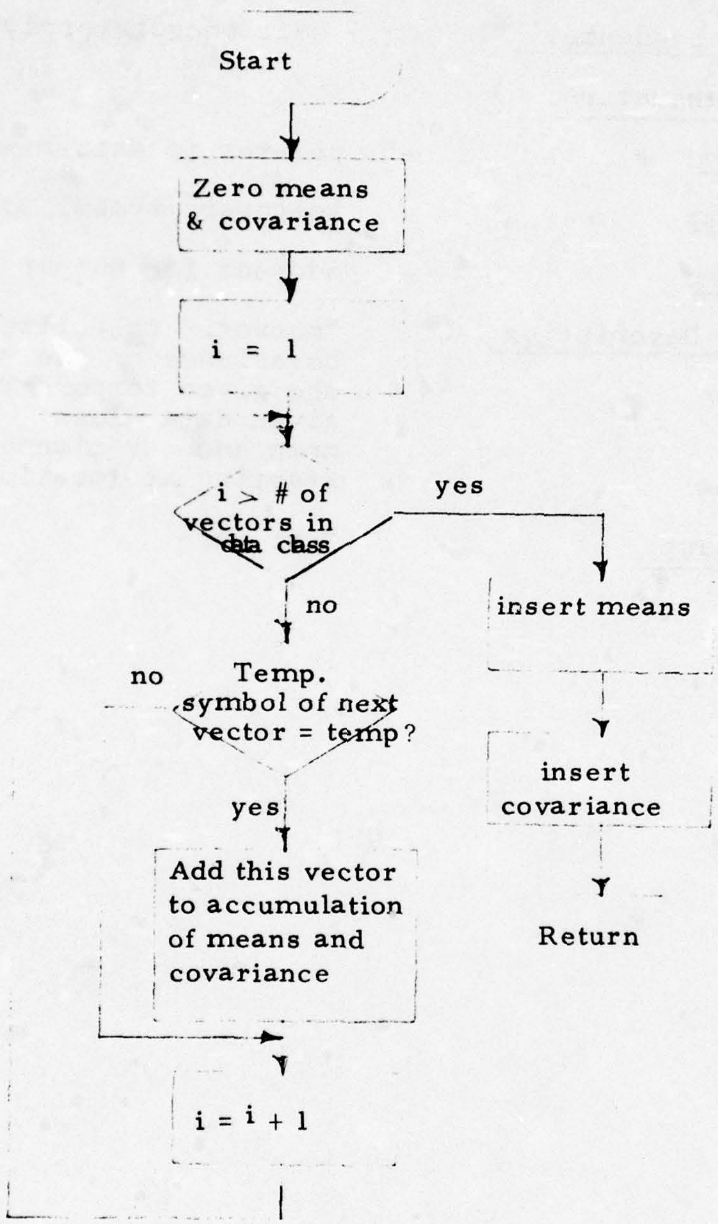
temp - temporary symbol to be used

ptro - address for output

Program Description: "mncvotr" calculates the mean and covariance of the vectors which have the given temporary symbol for the given data class. The resulting mean and covariance is stored starting at location ptro.

Flow Chart: Next page

mncvotr


Start
Zero means & covariance
i = 1
i > # of vectors in data class
yes
no
insert means
no
Temp. symbol of next vector = temp?
insert covariance
yes
Add this vector to accumulation of means and covariance
Return
i = i + 1

Internal Subroutine Name: mod1

Calling Sequence: call mod1 (lptr, field, pairptr, vptr, ndim, ptrs, nodeno)

Input Parameters:

lptr — pointer to top of "lbuff", the temporary file in the process directory which contains the desired modified logic

field — number of fisher thresholds used in evaluation

pairptr — pointer into the header information for the desired class pair in the logic block of the selected fisher node

vptr — pointer to the criteria logic block of the pair selected for modification in "pairmod"

ndim — dimensionality of data

ptrs — array of 5 pointers

ptrs(1) - pointer to the top of the "sysdata" file
ptrs(2) - pointer to the word 73 of the "scratch" file
ptrs(3) - pointer to the top of the "display"file
ptrs(4) - pointer to the top of the tree name file
ptrs(5) - pointer to the top of the logic file

nodeno — an array of 2 integers

node(1) - the node number of class A of the pair A/B
node(2) - the node number of class B of the pair A/B

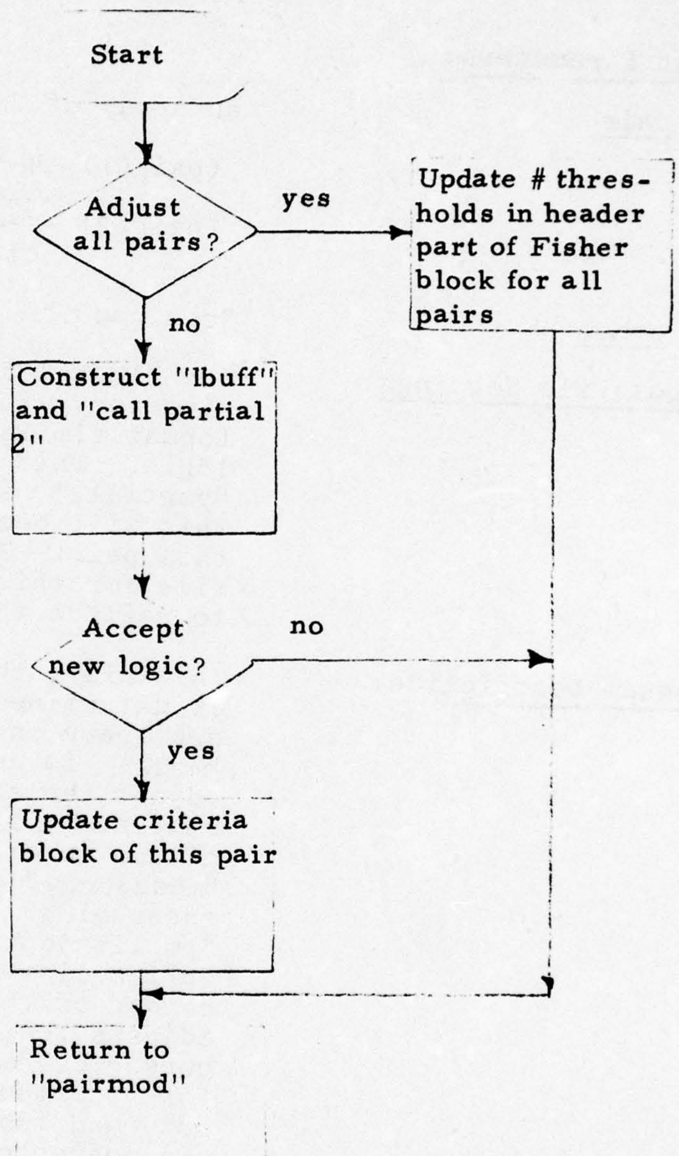
Output File Setting: The temporary logic block in the process directory, "lbuff" is built with the format similar to the one-space group logic block format.

Program Description: If the number of thresholds is to apply to all pairs, section LPRa of the logic block header for all pairs is replaced with the new number of thresholds and no "mini" confusion matrix, printed by subroutine "partial2", is generated. Control is returned to "pairmod".

Otherwise, the "lbuff" file is constructed under proper format and "partial2" is called. Upon acceptance of the modification, the section LPRa of the logic block header for this pair is updated. Then the program returns to "pairmod".

Flow Chart: See following page

modl


Start
Adjust all pairs?
yes
Update # thresholds in header part of Fisher block for all pairs
no
Construct "lbuff" and "call partial 2"
Accept new logic?
no
yes
Update criteria block of this pair
Return to "pairmod"

Internal Subroutine Name: mod2

Calling Sequence: call mod2 (ndim, ptrs, pair, pairptr, lptr, nodeno)

Input Parameters:

pair — an array of 2 class names

(pair(1) 4-character node name of class A in pair A/B

(pair(2) 4-character node name of class B in pair A/B

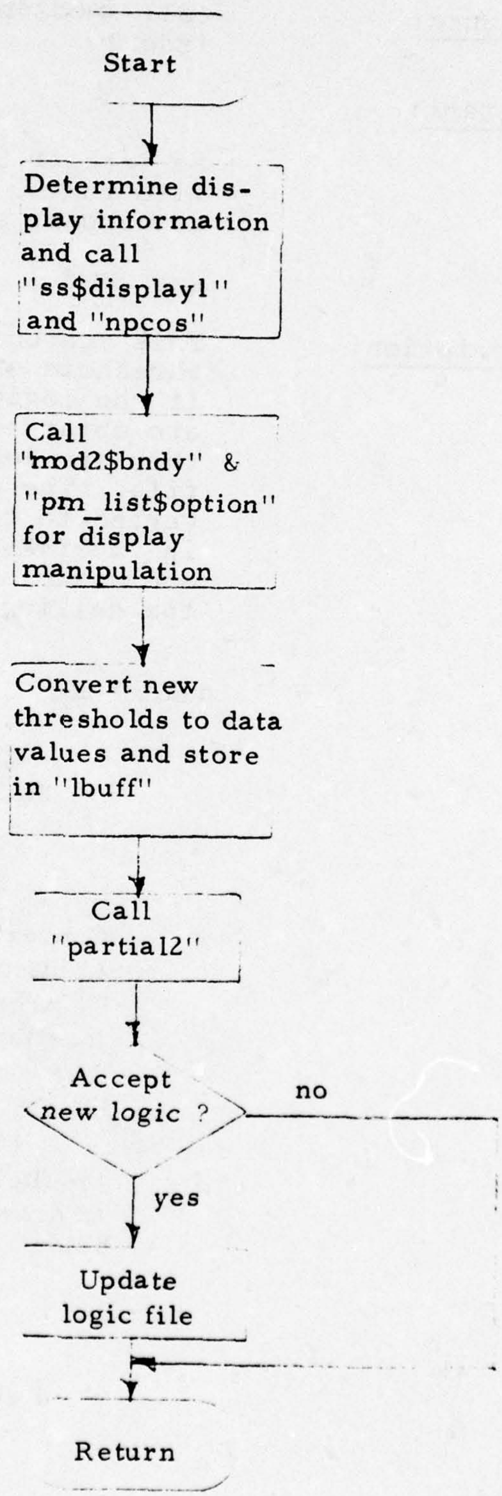
other — "see mod 1"

Output File Settings: The "lbuff" file in the process directory will contain logic in a format similar to one-space group logic. This file is evaluated by "partial2" and if the user desires, this will become the new logic for this pair. The section of the logic file for this node will be adjusted to reflect this new addition.

Program Description: Initially, the current type of logic is determined, either fisher or arbitrary one-space. The subroutine "mod2" is used in both instances to adjust thresholds. The moos subroutine "ss$display1" and "npcos" are called to display the histogram then "mod2$bndy" draws any existing thresholds. The subroutine "pm_list$option" writes the display option list in the upper-right hand corner of the screen. When the user adjusts the thresholds using the cursor (see the user's documentation for "pairmod"), the tektronix points returned from "multeks$read_xhair" are converted to data projection values and stored in "lbuff". The logic evaluation routine "partial2" is called to present a "mini" confusion matrix. Upon acceptance, the "lbuff" file is copied into the logic file and control is returned to "pairmod".

Flow Chart: See following page

mod2


Start
Determine display information and call "ss$display1" and "npcos"
Call "mod2$bndy" & "pm_list$option" for display manipulation
Convert new thresholds to data values and store in "lbuff"
Call "partial2"
Accept new logic ?
no
yes
Update logic file
Return

Internal Subroutine Name: mod2$bndy

Calling Sequence: call mod2$bndy (pairptr, ptrs, index)

Input Parameters:

index — an integer offset from top of logic file to the two-word header for the class pair selected

other — see mod 1

Program Description: This subroutine displays any existing threshold on the current histogram. If the logic is fisher, the thresholds are obtained from "lbuff", otherwise they are located in the "display" file. These thresholds are converted to tektronix points for use in "multeks$line". The thresholds are drawn and control is returned to the calling program.

Flow Chart:

mod2$bndy

Start

↓

Convert thresholds to Tektronix locations

↓

Call "multeks$line" to draw thresholds

↓

Return

Internal Subroutine Name: mod3

Calling Sequence: call mod3 (pairptr, ptrs, ndim, pair, lptr, nodeno)

Input Parameters:

pair — see "mod2"

other — see "mod1"

Output File Settings: The "lbuff" file will contain logic in a format similar to arbitrary one-space logic. The file is evaluated by "partial2" and if the user desires, this will become the new logic for this pair. The section of the logic file for this node will be adjusted to reflect this new addition.

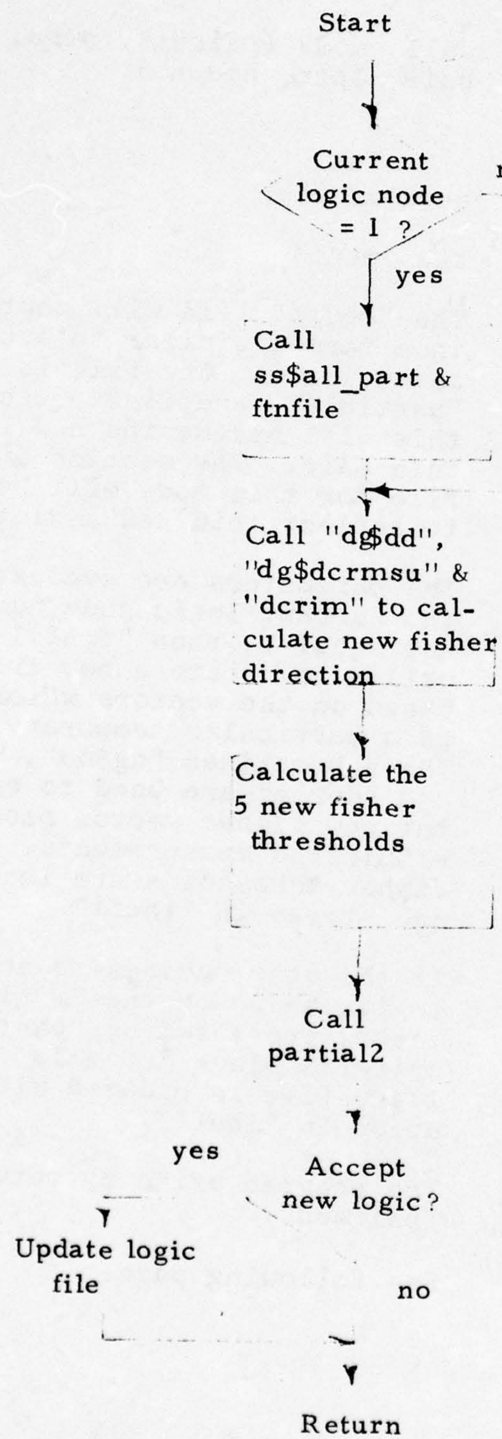
Program Description: The parameters are evaluated and if the current logic node number does not equal 1, then "ftnfile" is called to create a new treename file based on the vectors which correspond to a particular temporary symbol. The subroutines "dg$dd", "dg$dcrmsu" and "dcrim" are used to calculate the new fisher vector based upon the eliminated measurements. The new fisher thresholds are then calculated and stored in "lbuff".

If the user decides to accept the new logic, based on the "mini" confusion matrix presented by "partial2", the criteria block for this pair in the logic file is updated with the information in "lbuff".

The program exits by returning to "pairmod".

Flow Chart: **See following page.**

mod3


Start
Current logic node = 1 ?
no
yes
Call ss$all_part & ftnfile
Call "dg$dd", "dg$dcrmsu" & "dcrim" to calculate new fisher direction
Calculate the 5 new fisher thresholds
Call partial2
Accept new logic?
yes
Update logic file
no
Return

Internal Subroutine Name: mod5

Calling Sequence: call mod5 (pairptr, ptrs, lptr, ndim, nodeno)

Input Parameters: See **"modl"**

Output File Settings: The "lbuff" file will contain logic in a format similar to the one-space group. This file is evaluated by "partial2" and if the user desires, this will become the new logic for this pair. The section of the logic file for this node will be adjusted to reflect this new addition.

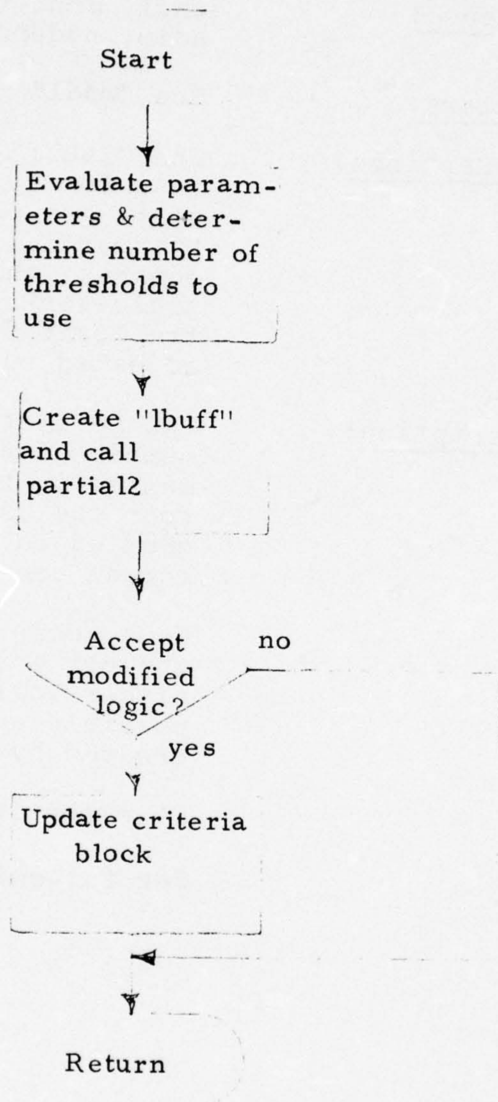
Program Description: The parameters are evaluated and the number of thresholds to be implemented in the evaluation is returned from the user. The "lbuff" file is created in the arbitrary one-space format and "partial2" is called.

Upon acceptance of the new logic, the criteria block is altered to fisher logic. The index to a possible auxiliary criteria block created by "pairmod" is deleted.

This routine returns to "pairmod".

Flow Chart: See following page

mod5


Start
Evaluate parameters & determine number of thresholds to use
Create "lbuff" and call partial2
Accept modified logic?
no
yes
Update criteria block
Return

Internal Subroutine Name: mod6

Calling Sequence: call mod6 (ptrs, lptr, ndim, pairptr, pair, nodeno)

Input Parameters:

pair: see "mod2"

other: see "mod1"

Output File Settings: The "lbuff" file will contain logic in a format similar to the one-space group. This file is evaluated by "partial2" and, if the user desires, this will become the new logic for this pair. The section of the logic file for this node will be adjusted to reflect this new addition.

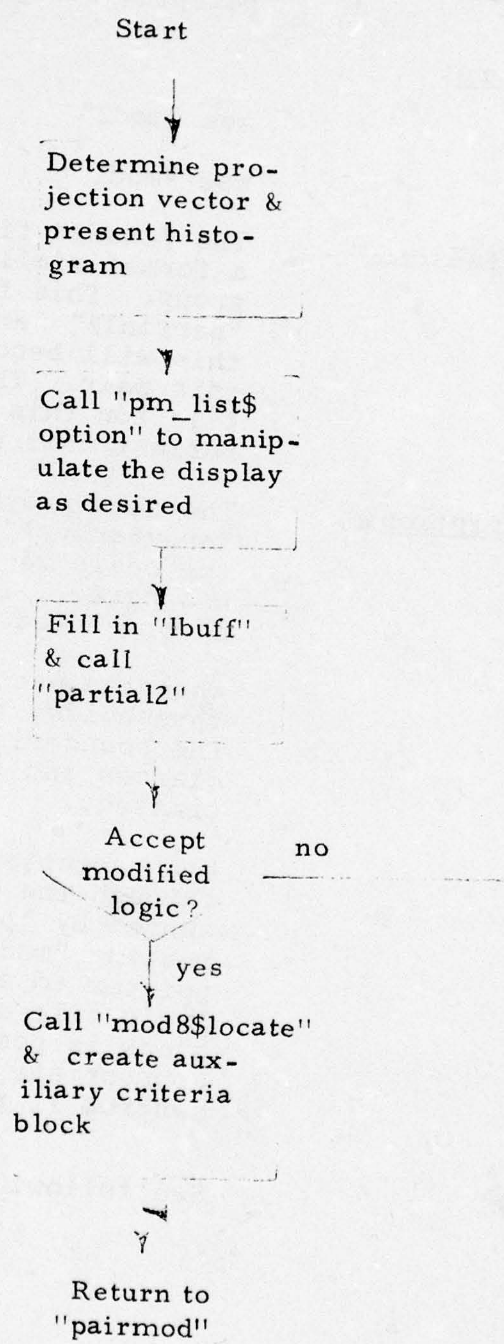
Program Description: The data projection subroutine "arbv$arbvcl" is called to determine the basis vector and display the histogram. The display option list is presented by "pm_list$option".

When the user draws the desired thresholds, the "convex" sides of the boundaries are associated with classes and the "lbuff" file is created.

Upon acceptance of the new logic, through the "mini"-evaluation presented by "partial2" the subroutine "mod8$locate" returns a pointer to a vacant block in the logic file and the auxiliary criteria block is positioned there. The appropriate links to it are made and control is returned to "pairmod".

Flow Chart: See following page

mod6


Start
Determine projection vector & present histogram
Call "pm_list$option" to manipulate the display as desired
Fill in "lbuff" & call "partial2"
Accept modified logic?
no
yes
Call "mod8$locate" & create auxiliary criteria block
Return to "pairmod"

Internal Subroutine Name: mod7

Calling Sequence: call mod7 (ndim, pairptr, pair, ptrs, names, nodeno, lptr)

Input Parameters:

pair see "mod2"

names an array of 2 characters

names(1) display symbol of class A in pair A/B
Names(2) display symbol of class B in pair A/B

other

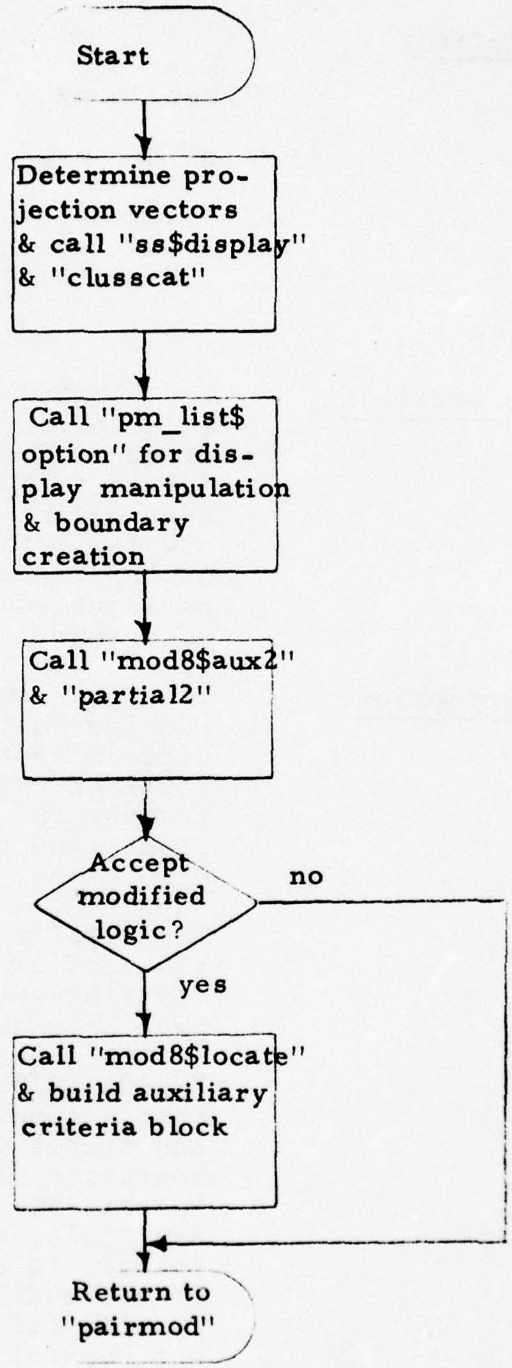
Output File Settings: The "lbuff" file will contain logic in a format similar to the two-space group logic format. The file is evaluated by "partial2" and if the user desires, this will become the new logic for the pair. The section of the logic file for this node will be adjusted to reflect this new addition.

Program Description: This program is the "pairmod" subroutine that creates or modifies discriminant plane logic. The plot routines "ss$display" and "clusscat" present the data projected on the fisher and fisher-orthonormal directions.

The display can be manipulated through the list of options presented by "pm_list$option". After the boundaries have been drawn and classes assigned to the convex region of each boundary, "mod8$aux2" converts these boundaries into the two-space group logic format and stores them in "lbuff". The "mini" confusion matrix is printed by "partial2". Upon acceptance of this new logic, mod8$locate" returns a pointer to a start in the logic file of the auxiliary criteria block. The "lbuff" file is copied into this area and control is returned to "pairmod".

Flow Chart: See following page

mod7


Start
Determine projection vectors & call "ss$display" & "clusscat"
Call "pm_list$option" for display manipulation & boundary creation
Call "mod8$aux2" & "partial2"
Accept modified logic?
no
yes
Call "mod8$locate" & build auxiliary criteria block
Return to "pairmod"

Internal Subroutine Name: mod8

Calling Sequence: call mod8 (pairptr, ptrs, ndim, nodeno, names, lptr)

Input Parameters:

names — see "mod7"

other — see "mod1"

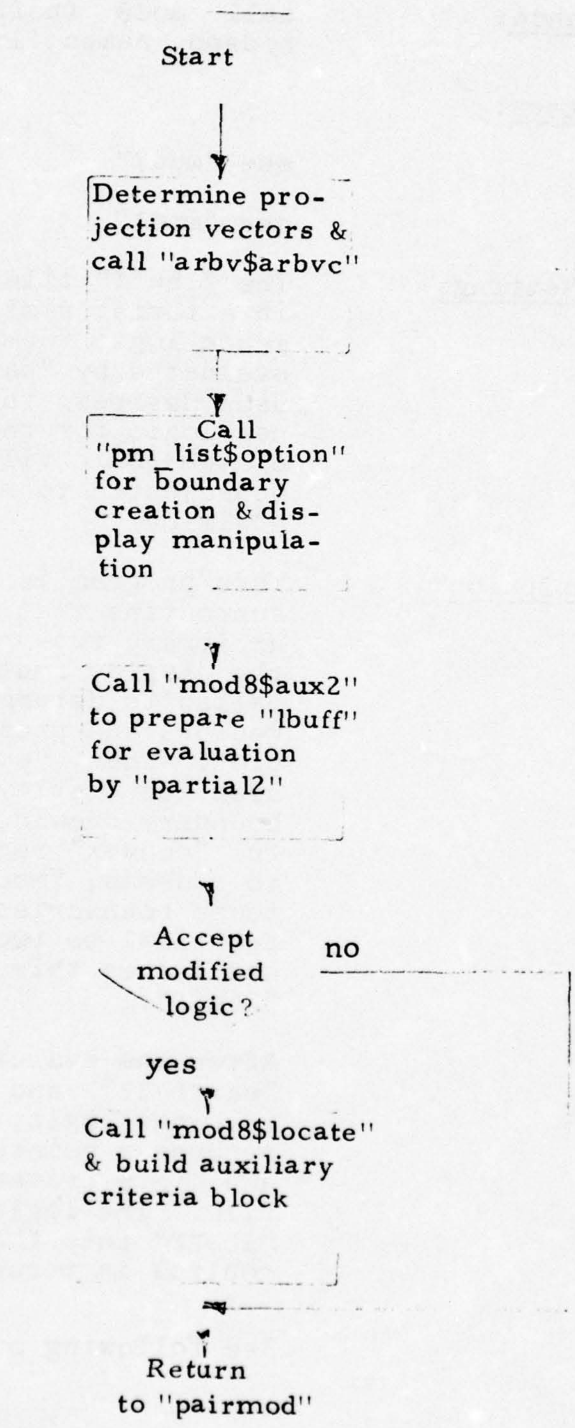
Output File Settings: The "lbuff" file will contain logic in a format similar to the two-space logic format. The file is evaluated by "partial2" and if the user desires, this will become the new logic for the pair. The section of the logic file for this node will be adjusted to reflect this new addition.

Program Description: This program is the "pairmod" subroutine that creates or modifies arbitrary two-space logic. Initially the display routine "arbv$arbvc" is called to determine the projection vectors and present the two-space plot. Then, "pm_list$option" is used for display manipulation and boundary drawing. After assigning the "convex" regions of the boundaries to classes, "mod8$aux2" converts these boundaries into a format identical to two-space group logic and stores this information in "lbuff".

After the evaluation is presented by "partial2" and if the user accepts this new logic, "mod8$locate" returns a pointer to the newly created auxiliary criteria block in the logic file. The logic is then copied from "lbuff" into the auxiliary block and control is returned to "pairmod".

Flow Chart: See following page

mod8


Start
Determine projection vectors & call "arbv$arbvc"
Call "pm_list$option" for boundary creation & display manipulation
Call "mod8$aux2" to prepare "lbuff" for evaluation by "partial2"
Accept modified logic?
no
yes
Call "mod8$locate" & build auxiliary criteria block
Return to "pairmod"

Internal Subroutine Name: mod8$aux2

Calling Sequence: call mod8$aux2(ptrs, lptr, ndim, aux_length)"

Input Parameters: See "mod1"

Output Parameters:

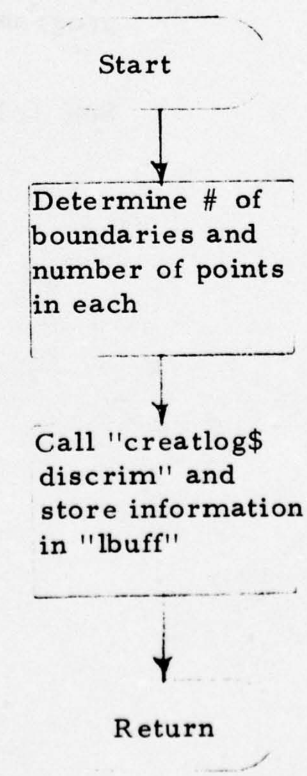
aux_length — length of auxiliary criteria block

Output File Settings: The "lbuff" will contain logic in the format similar to the two-space group logic format

Program Description: The boundary points and convex points are determined from the "display" file and "creatlog$discrim" is called to convert them to line segments and discriminants. This information is stored in the two-space group logic format in the "lbuff" file.

Flow Chart:

mod8$aux2



Internal Subroutine Name: mod8$locate

Calling Sequence: call mod8$locate (aptr, pairptr, ndim, length)

Input Parameters:

length — length of auxiliary criteria block

other — see "modl"

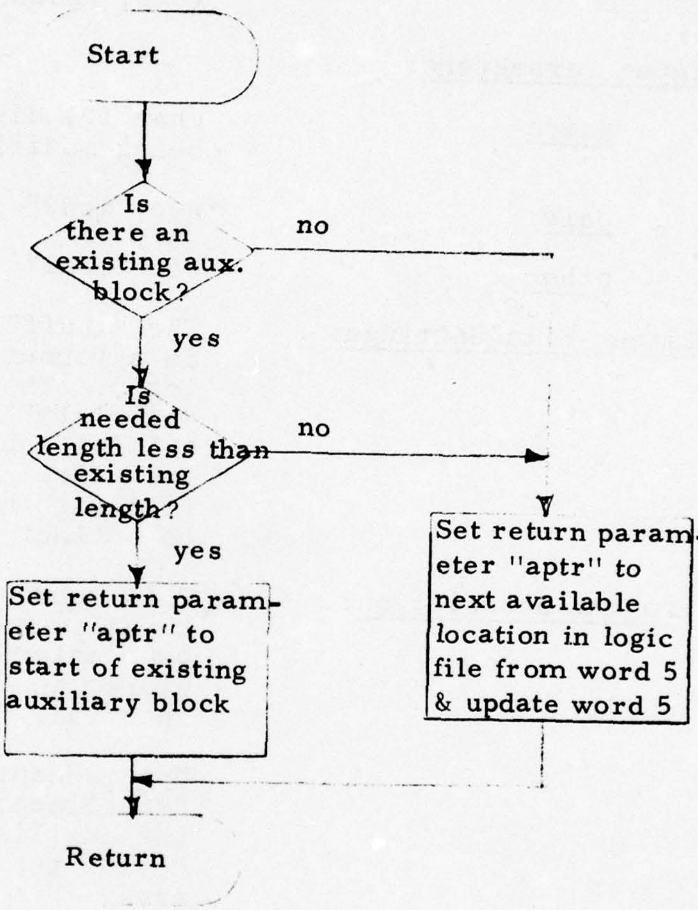
Output Parameters:

aptr — pointer to start of auxiliary criteria block in logic file

Program Description: The parameter "length" is compared against the length of any existing auxiliary criteria block. If "length" is less than this number, "aptr" is returned to the current location of this block. Otherwise, the pointer value is taken from word 5 of the logic file, the index to the next free location in the logic file. Word 5 is reset and control is returned to the calling program.

Flow Chart: See following page

mod8$locate



Start
Is there an existing aux. block?
no
yes
Is needed length less than existing length?
no
yes
Set return parameter "aptr" to start of existing auxiliary block
Set return parameter "aptr" to next available location in logic file from word 5 & update word 5
Return

Internal Subroutine Name: mod9

Calling Sequence: call mod9 (ptrs, pairptr, pair, lptr, nodeno, pairc)

Input Parameters:

pairc — char (2) display symbols of classes being modified

pair — see "mod2"

other — see "mod1"

Output File Settings: The "lbuff" file will contain logic in a format similar to Boolean logic. The file is evaluated by "partial2" and, if the user desires, this will become the new logic for the pair. The section of the logic file for this node will be adjusted to reflect this new addition.

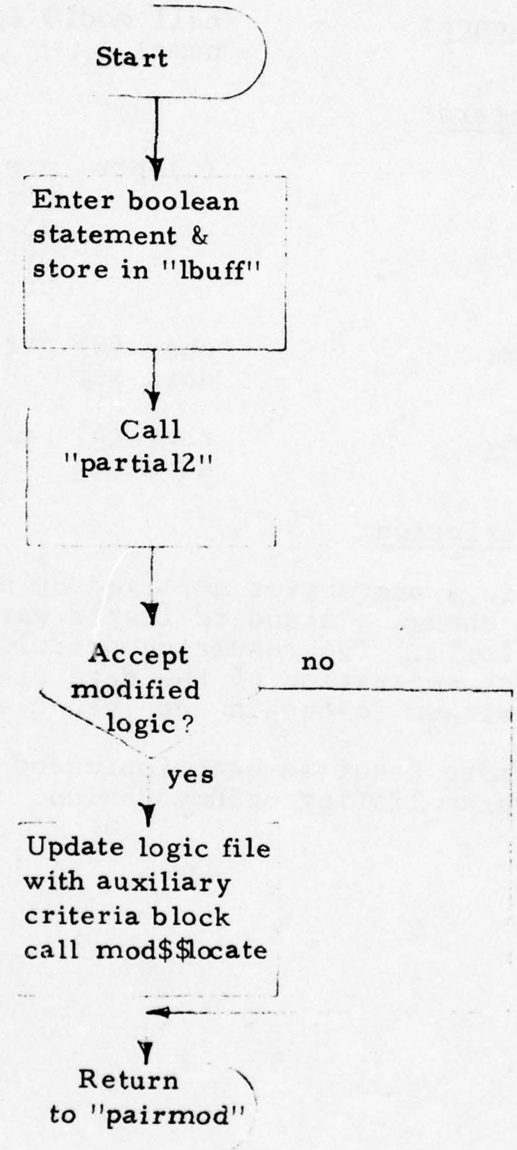
Program Description: This subroutine is used by "pairmod" to allow for Boolean logic. The user enters the statement and it is stored in "lbuff" for use by "partial2."

Upon acceptance of this logic, "mod8$locate" returns a pointer to the auxiliary criteria block and copies the "lbuff" file into this area.

"mod9" then returns to "pairmod."

Flow Chart: See following page.

mod9


Start
Enter boolean statement & store in "lbuff"
Call "partial2"
Accept modified logic?
no
yes
Update logic file with auxiliary criteria block call mod$$locate
Return to "pairmod"

Internal Subroutine Name: modl0

Calling Sequence: call modl0 (ptrs, treename, nodename)

Input Parameters:

ptrs — (5) ptr
- ptrs(1) - sysdata
- ptrs(2) - scratch
- ptrs(3) - display
- ptrs(4) - treename
- ptrs(5) - mooslogic

treename — char (8) tree name of the current data set

nodename — char (4) class name of the current data set

Program Description:

modl0 is a subroutine utilized by pairmod which allows the user to change a standard fisher pairwise logic into a group type logic. The resulting pairwise logic does not produce complete separation of the data classes. Further logic must be developed to obtain complete classification.

For a more detailed description of the operation of modl0, see the program listing documentation.

Internal Subroutine Name: moosinitiate

Calling Sequence: call moosinitiate

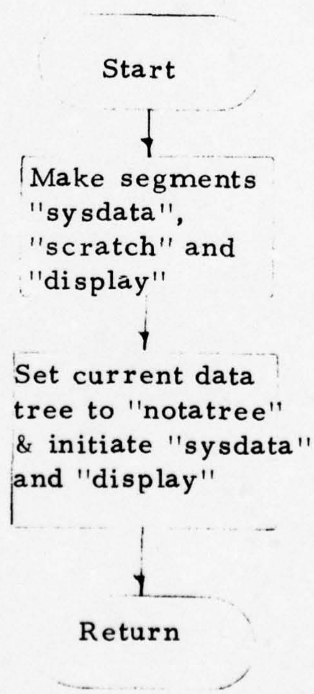
Output File Settings:

"sysdata" file
CSS1 = "notatree"
CSS2 = "nono"
CSS4 = 1
CSS3/CSS5/CSS6 = 0
All F1 entries = "notatree"
the first S1 entry = "nono"

"display" file
word 1 = 0

Program Description: "moosinitiate" initializes files "sysdata", "scratch" and "display" in the user's temporary directory and sets initial values into the sysdata and display files. This routine is called upon user entrance to the system by "ut$ckparam".

Flow Chart:



Utility Function Name: moosmode

Calling Sequence: Type in "moosmode"

Program Description:

moosmode lists all trees whose dimensionality is greater than 100, and all trees that have 100 or less dimensions and no treename files. The user may have treename files created for any or all of the trees in the second list. Treename files are created through calls to moosmode$ctreename.

If there are no trees with greater than 100 dimensions, and all trees have treename files, moosmode sets sense switch 3 to "off," thus leaving the excess measurement mode and allowing normal MOOS functions to be executed.

For a more detailed description of the operation of moosmode, see the program listing documentation.

For a list of programs which may be executed while the system is in the excess measurement mode, see Sections 1 and 4.2.1.

Internal Subroutine Name: moosmode$ctreename

Calling Sequence: call moosmode$ctreename (t, treename)

Input Parameters:

t — fixed (35) tree number (same as no. which would appear in CSS3 in sysdata)

treename — char (8) tree name of the tree whose "treename file" is being created.

Program Description:

ctreename creates a treename file for a one-(1-) level data tree. Data class files, and all sysdata information must be correctly set up beforehand.

Internal Subroutine Name: msxform

Calling Sequence: call msxform(p1, ndim1, p2, ndim2)

Input Parameters:

p1 - a pointer to a vector to be transformed

ndim1 - dimension of vector to be transformed

p2 - pointer to a place where transformed vector can be placed

ndim2 - dimension of transformed vector

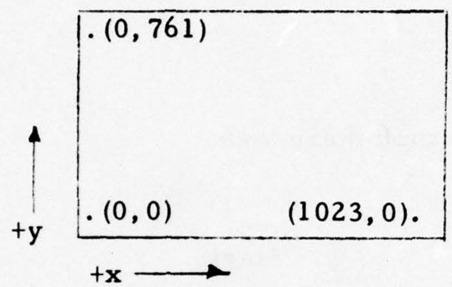
Program Description: "msxform" is a routine created by routine "measxform". Its purpose is to transform a vector according to a set of entered transformation rules. However, those rules are not known until routine "measxfrm" is run and therefore no flow chart can be written for "msxform".

Internal Subroutine: multeks

"multeks" is a set of 6 PL/1 subroutines that can be used by the programmer to control the Tektronix 4002A under control of MULTICS.

Points on the display area of the storage tube range from 0 - 1023 in the x direction and 0 - 761 in the y direction. Point (0,0) on the screen is located at the bottom left hand corner of the screen.

4002A



In the description of the "multeks" subroutines, a tekpoint refers to an (x,y) point in this range.

The 6 subroutines are: (1) multeks$erase

(2) multeks$home

(3) multeks$print_char

(4) multeks$position_ptr

(5) multeks$line

(6) multeks$read_xhair

Internal Subroutine Name: multeks$erase

Calling Sequence: call multeks$erase

Program Description: "multeks$erase" subprogram erases the screen and places the terminal in alphanumeric mode. The cursor is not moved from the current cursor tekpoint.

Flow Chart:

multeks$erase

Start

Output the CAN control character to erase the screen

Delay .5 seconds for screen to erase

Output US control character to return to alphanumeric mode

Return

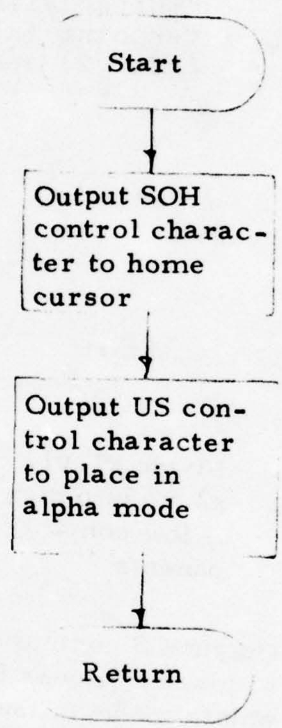
Internal Subroutine Name: multeks$home

Calling Sequence: call multeks$home

Program Description: "multeks$home" positions the cursor at tekpoint (0,761), and places the terminal in alphanumeric mode.

Flow Chart:

multeks$home



Internal Subroutine Name: multeks$line

Calling Sequence: call multeks$line (x1, y1, x2, y2)

Input Parameters:

x1, y1 — the first (x,y) tekpoint [each fixed bin]

x2, y2 — the second (x,y) tekpoint [each fixed bin]

Program Description: "multeks$line" draws a vector from tekpoint (x1,y1) to tekpoint (x2, y2) and returns to alpha mode.

Flow Chart:

multeks$line

Start

↓

Divide x1, y1, x2, y2 into high & low components

↓

Output GS control char. to place in linear interpolate mode, followed by high y1, low y1, high x1, low x1, high y2, low y2, high x2, low x2 to draw vector, followed by US control char. to return to alpha

↓

Return

Internal Subroutine Name: multeks$position_ptr

Calling Sequence: call multeks$position_ptr(x,y)

Input Parameters:

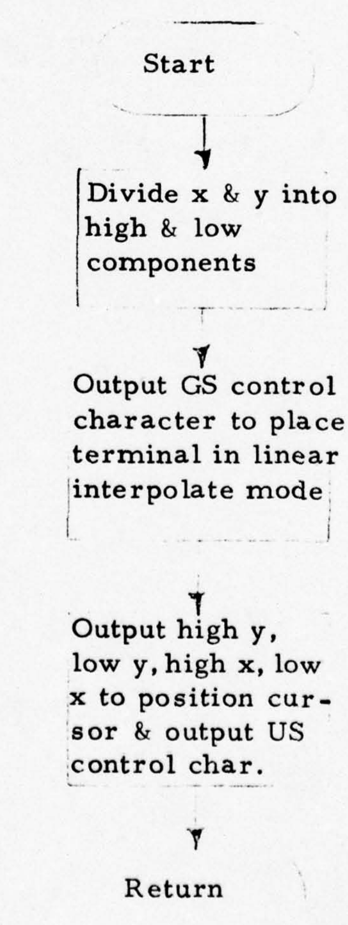
x — the x tekpoint [fixed bin]

y — the y tekpoint [fixed bin]

Program Description: "multeks$position_ptr" positions the cursor at the given (x,y) tekpoint and returns to alphanumeric mode.

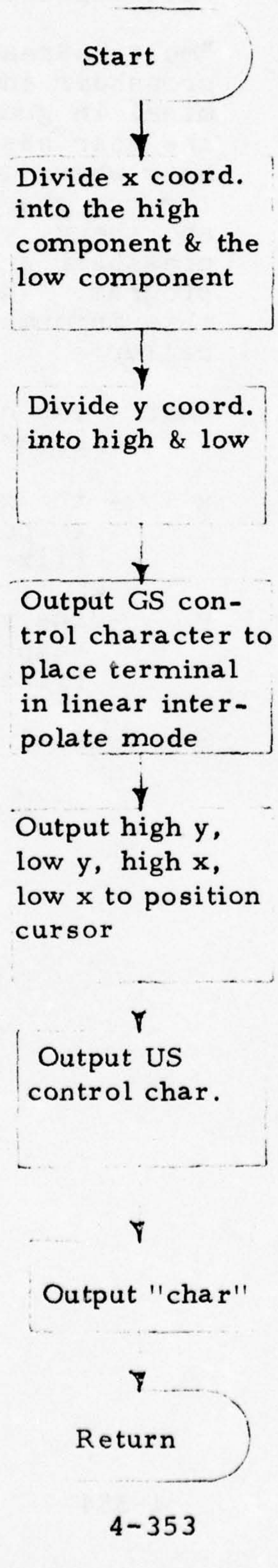
Flow Chart:

multeks$position_ptr



Internal Subroutine Name:	multeks$print_char
Calling Sequence:	call multeks$print_char (char,x,y)
Input Parameters:	
char	the character to be printed [char (1)]
x	the x tekpoint coordinate where "char" is to be placed [fixed bin)]
y	the y tekpoint where "char" is to be placed [fixed bin]
Program Description:	"multeks$print_char" positions a character at a given (x,y) tekpoint and places the terminal in alphanumeric mode.
Flow Chart:	Next page.

multeks$print_char



Internal Subroutine Name: multeks$read_xhair

Calling Sequence: call multeks$read_xhair (char, x, y)

Program Description: "multeks$read_xhair" turns on the crosshair and thus places the terminal in graphics input mode. When the user has positioned the crosshair where desired, he may then enter any character. This character and the x, y coordinates of the crosshair are transmitted to the program. The program then decodes this information and returns to the caller:

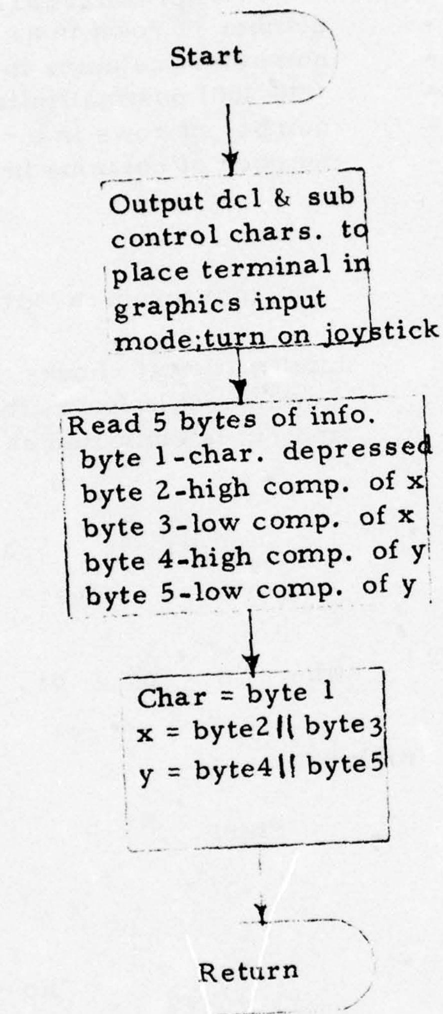
char - the character depressed [char (1)]

x - the x-coordinate of the tekpoint of the crosshair [fixed bin]

y - the y-coordinate of the tekpoint of the crosshair [fixed bin]

Flow Chart: See following page.

multeks$read_xhair



Internal Subroutine Name: multmat

Calling Sequence: call multmat (a, ai, aj, b, bi, bj, c)

Input Parameters:

- a - (100, 100) premultiplier
- ai - number of rows in a.
- aj - number of columns in a.
- b - (100, 100) postmultiplier.
- bi - number of rows in b.
- bj - number of columns in b.

Output Parameter:

- c - (100, 100) product matrix

Program Description: multmat first checks that the matrices are conformable for multiplication. The product matrix is computed as

$$c(ij) = \sum_{k=1}^{n} a(i,k) \times b(k,j)$$

where $n = aj = bi$.

Flow Chart:

multmat

Start

aj = bi? — no

yes

Compute
Product
Matrix C

Return

MOOS Function Name: nlm

MOOS Function Number: 202

Calling Sequence: Type in "nlm [(treename) (nodename) "

Input Parameters: Standard optional data set selection parameters

Output File Settings: nlm sets up the display file for a two-space plot of the selected (or the clustered) data set.

Program Description: nlm checks the total number of vectors in the selected data set. If this number exceeds 200, internal subroutine clusterl is called which creates a new tree with fewer vectors for the use of nlm.

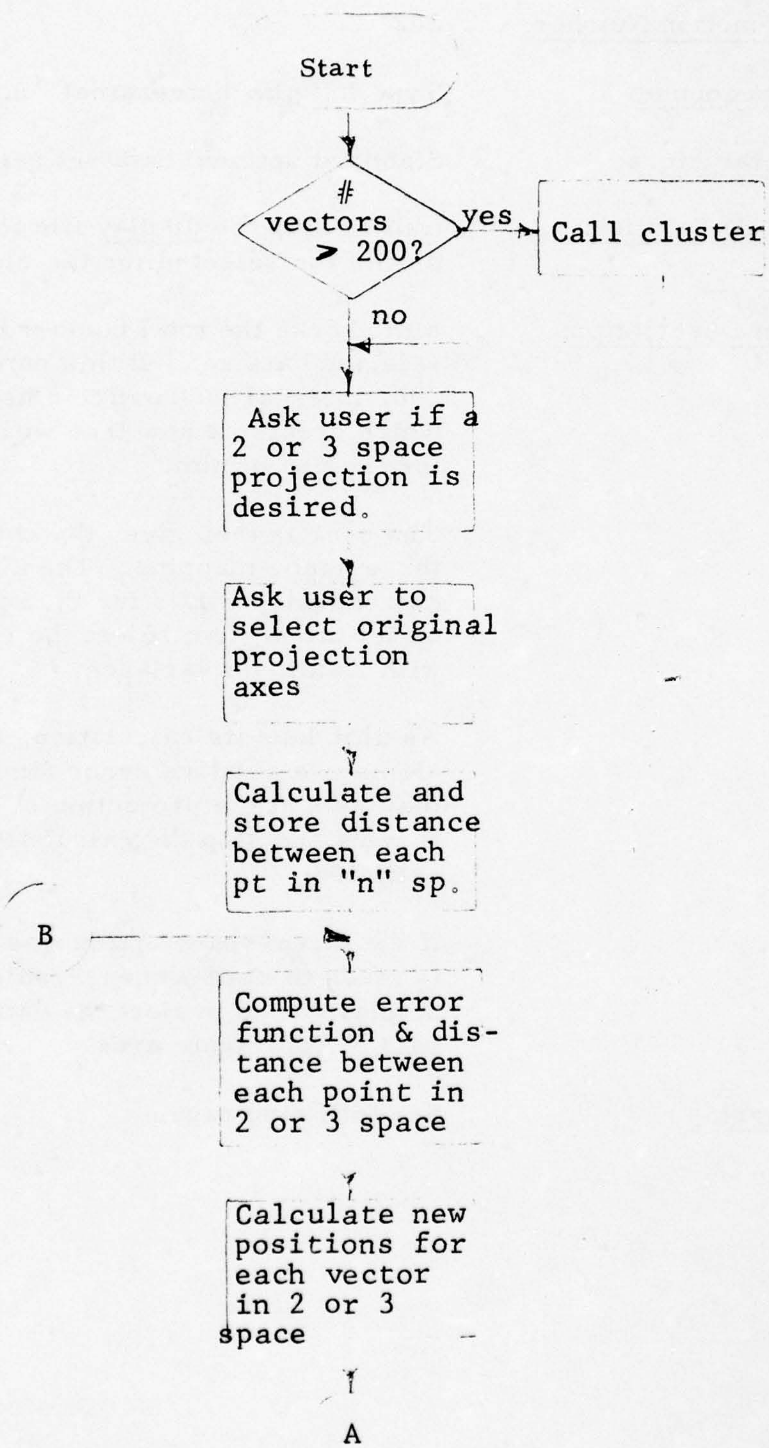
The user is then given the choice of a two or three space mapping. The user can select any coordinate axis for the initial projection, or let the system select the coordinate axes with maximum variance.

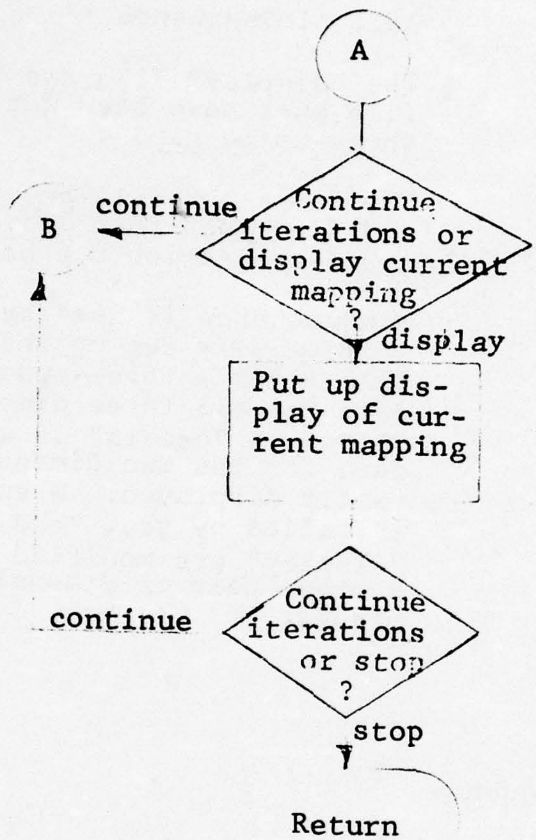
As nlm does its calculation, it computes and displays a relative error function. The user may look at the projection of his data at intervals and stop the calculation when he is satisfied.

If the three-space option was chosen, a call is made to nlm$sequence which sets up the display file to project the data on the first pair of coordinate axes.

Flow Chart: See following page.

nlm


Start
#
vectors
> 200?
yes
Call cluster
no
Ask user if a 2 or 3 space projection is desired.
Ask user to select original projection axes
Calculate and store distance between each pt in "n" sp.
B
Compute error function & distance between each point in 2 or 3 space
Calculate new positions for each vector in 2 or 3 space
A


A
continue
B
Continue iterations or display current mapping ?
display
Put up display of current mapping
continue
Continue iterations or stop ?
stop
Return

Internal Subroutine Name: nlm$sequence

Calling Sequence: call nlm$sequence

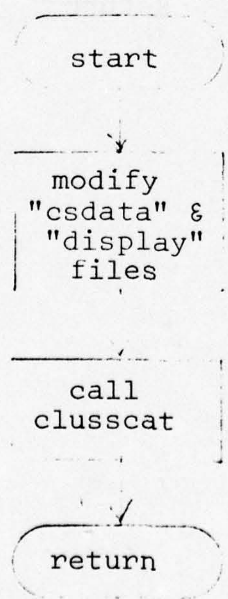
Input File Settings: The "display" file and the "csdata" file must have been set up for a three-space nlm.

Output File Settings: The "display" file and "csdata" modified such that a different two-space projection may be displayed.

Program Description: nlm$sequence is used by nlm to initially set up the "display" file after a three-space nlm. The data for all three dimensions are stored in "csdata" as well as the data for the two dimensions currently displayed. When nlm$sequence is called by seq, "csdata" and "display" are modified such that another pair of dimensions is displayed.

Flow Chart:

nlm$sequence



Internal Subroutine Name: nmv_logic

Calling Sequence: call nmv_logic (cn, logptr, vectptr, ndim, eptr)

Input Parameters:

cn	current logic node number
logptr	pointer to first word of logic block
vectptr	pointer to first measurement of vectors
ndim	number of dimensions
eptr	pointer to area in which error information will be placed

Output Parameters:

cn	logic node number of assigned class
eptr	pointer to next available slot in error file

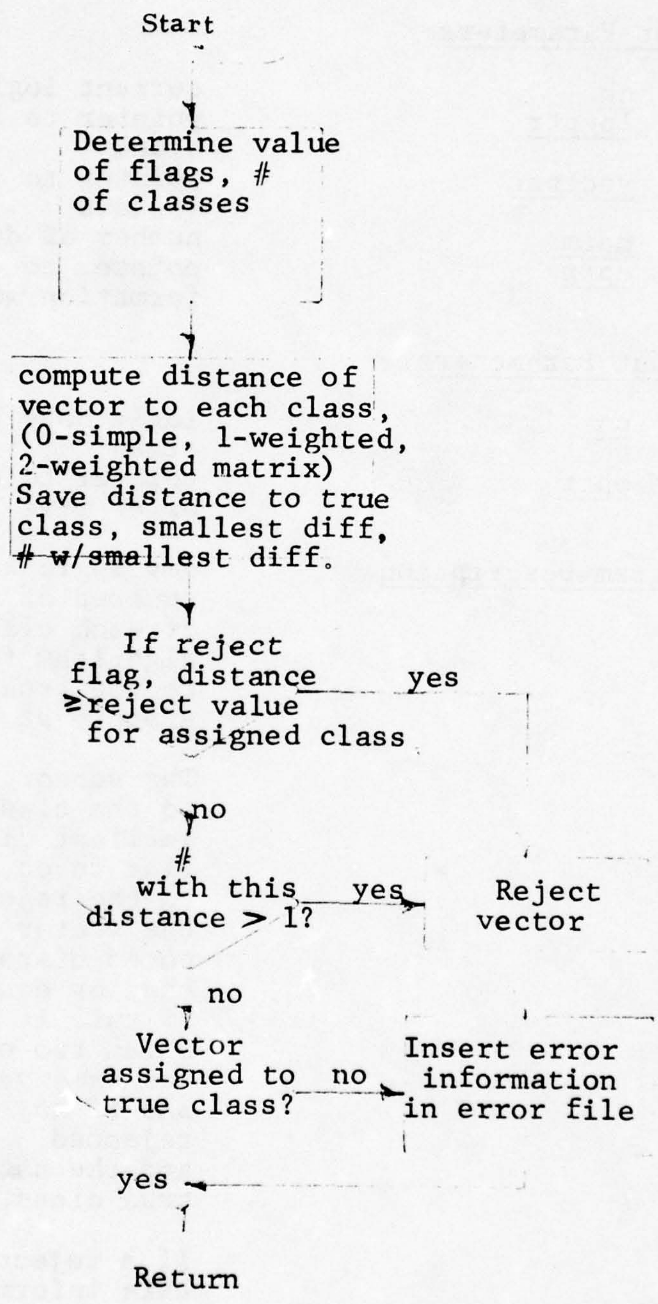
Program Description: nmv_logic computes the distance squared of the vector to the means of each class according to the algorithm flagged. The distance to the true class is stored separately at this time.

The vector is temporarily assigned to the class which yields the smallest distance. If the reject flag is on, this value is compared to the reject value for that class; the vector is rejected if the computed distance squared is greater than or equal to the reject value. If this is not the case, ties in which two or more classes yielded the same value are checked for, and if any exist, the vector is rejected. If this is not the case, and the assigned class is not the true class, an error is generated.

If a reject or error has occurred, this information is inserted in the error file. An output distance value is obtained by taking the square root of the distance squared.

Flow Chart: See following page.

nmv_logic


Start
Determine value of flags, # of classes
compute distance of vector to each class, (0-simple, 1-weighted, 2-weighted matrix) Save distance to true class, smallest diff, # w/smallest diff.
If reject flag, distance ≥ reject value for assigned class
yes
no
with this distance > 1?
yes
Reject vector
no
Vector assigned to true class?
no
Insert error information in error file
yes
Return

MOOS Function Name: nmv

MOOS Function Number: 63

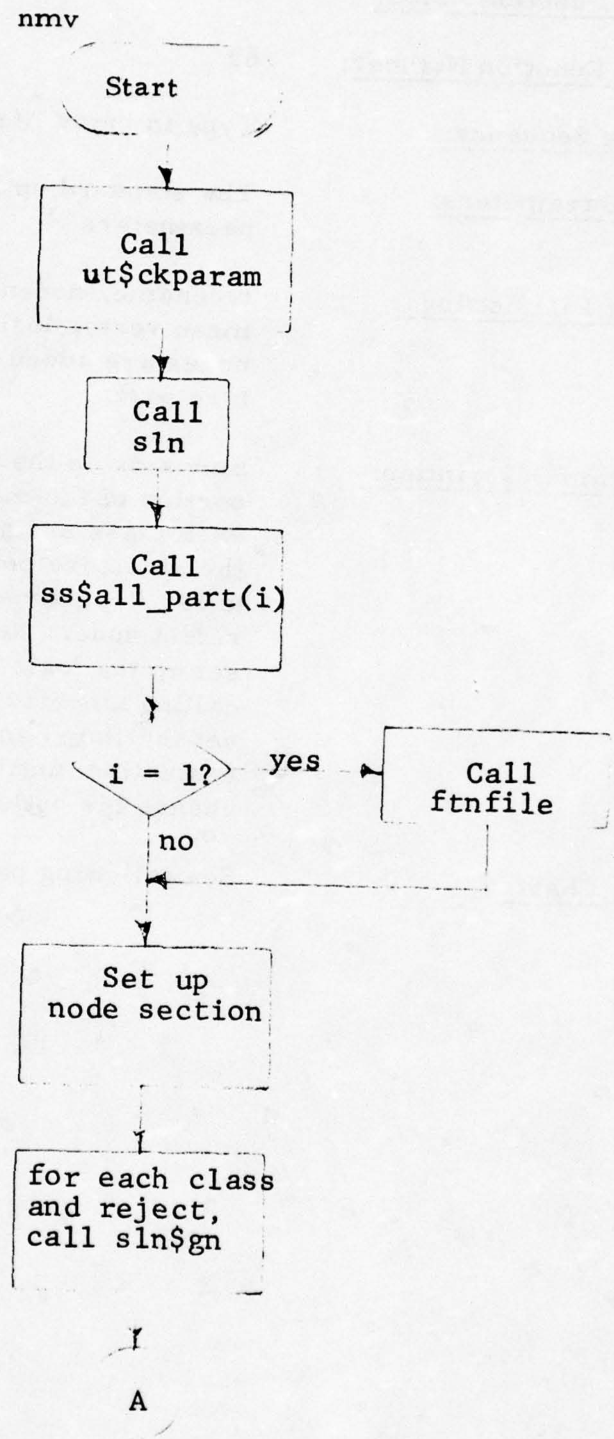
Calling Sequence: Type in "nmv [(treename) [(nodename) "

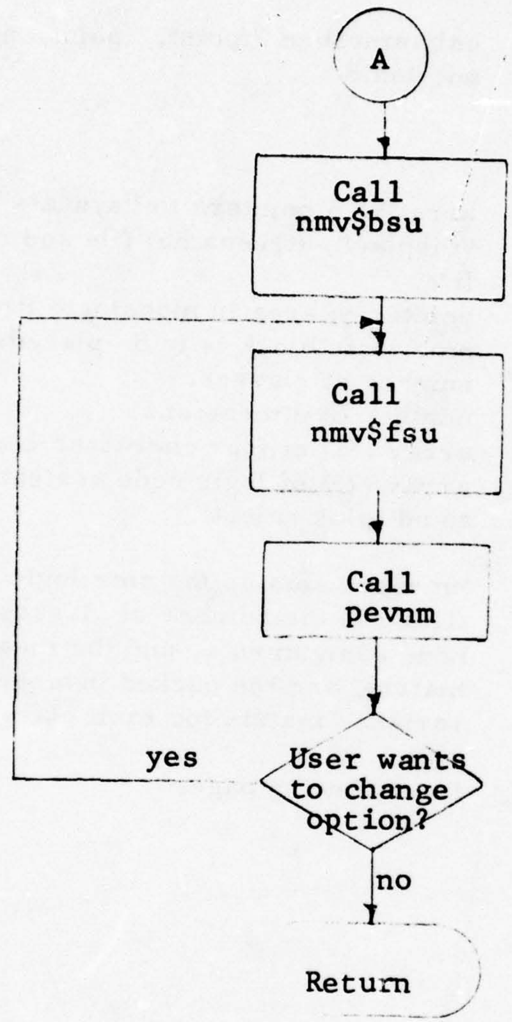
Input Parameters: The standard optional data set selection parameters

Output File Settings: treename, nodename "logic" file: a nearest mean vector logic block is added, and ncls + 2 nodes are added to the node part of the structure part.

Program Description: nmv sets up the node section of the structure portion of the mooslogic file, calls sln$gn for each class assignment of a node section of the structure portion for each lowest node under (treename), (nodename) and for the reject node. Next, nmv calls nmv$bsu to set up the logic block, and then cycles through calling nmv$fsu (to input the user options and set the flags) and calling pevnm (for partial evaluation) until the user does not wish to change his options.

Flow Chart: See following page.


nmv
Start
Call ut$ckparam
Call sln
Call ss$all_part(i)
i = 1?
yes
Call ftnfile
no
Set up node section
for each class and reject, call sln$gn
A


A
Call
nmv$bsu
Call
nmv$fsu
Call
pevnm
yes
User wants
to change
option?
no
Return

Internal Subroutine Name: nmv$bsu

Calling Sequence: call nmv$bsu (fpoint, lpoint, ncls, ndim, nd, lnd)''

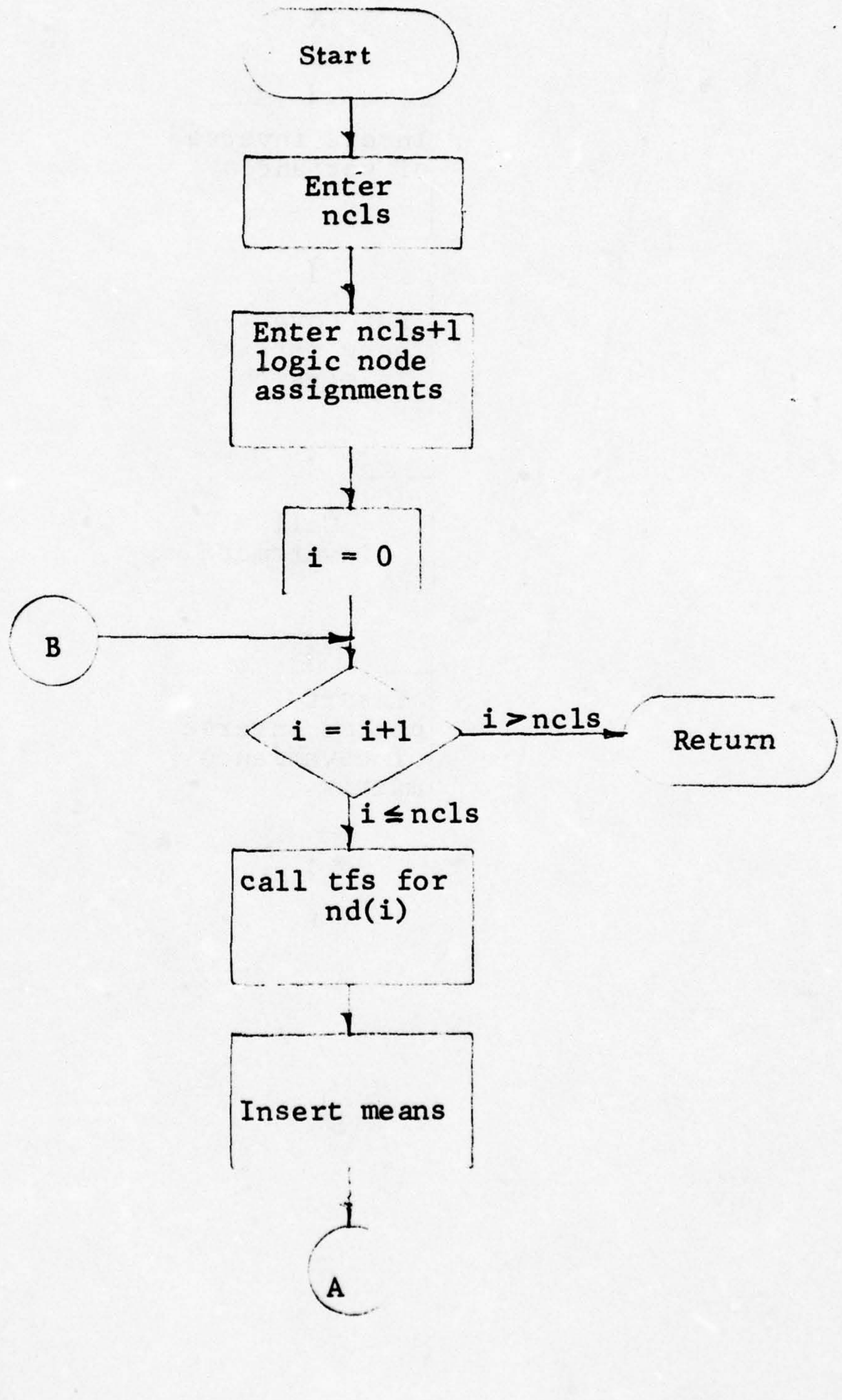
Input Parameters:

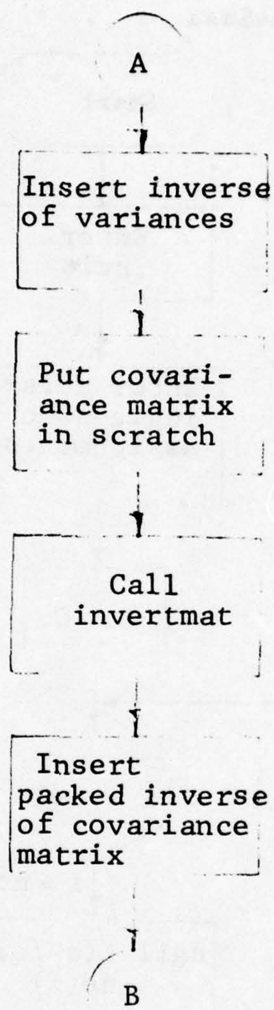
fpoint	-	array of 5 pointers to ''sysdata'', ''scratch'', ''display'', (treename) file and mooslogic file.
lpoint	-	pointer to area in mooslogic file in which the nmv logic block is to be placed.
ncls	-	number of classes.
ndim	-	number of dimensions.
nd	-	array (72) of four character class names.
lnd	-	array (73) of logic node assignments parallel to nd (plus reject).

Program Description: nmv$bsu sets up the nmv logic block by filling in the number of classes, the logic node assignments, and the means, weighting matrix, and the packed inverse of the covariance matrix for each class.

Flow Chart: See following page.

nmv$bsu


Start
Enter
ncls
Enter ncls+1
logic node
assignments
i = 0
B
i = i+1
i > ncls
Return
i ≤ ncls
call tfs for
nd(i)
Insert means
A


A
Insert inverse of variances
Put covariance matrix in scratch
Call invertmat
Insert packed inverse of covariance matrix
B

Internal Subroutine Name: nmv$fsu

Calling Sequence: call nmv$fsu (point, scptr)

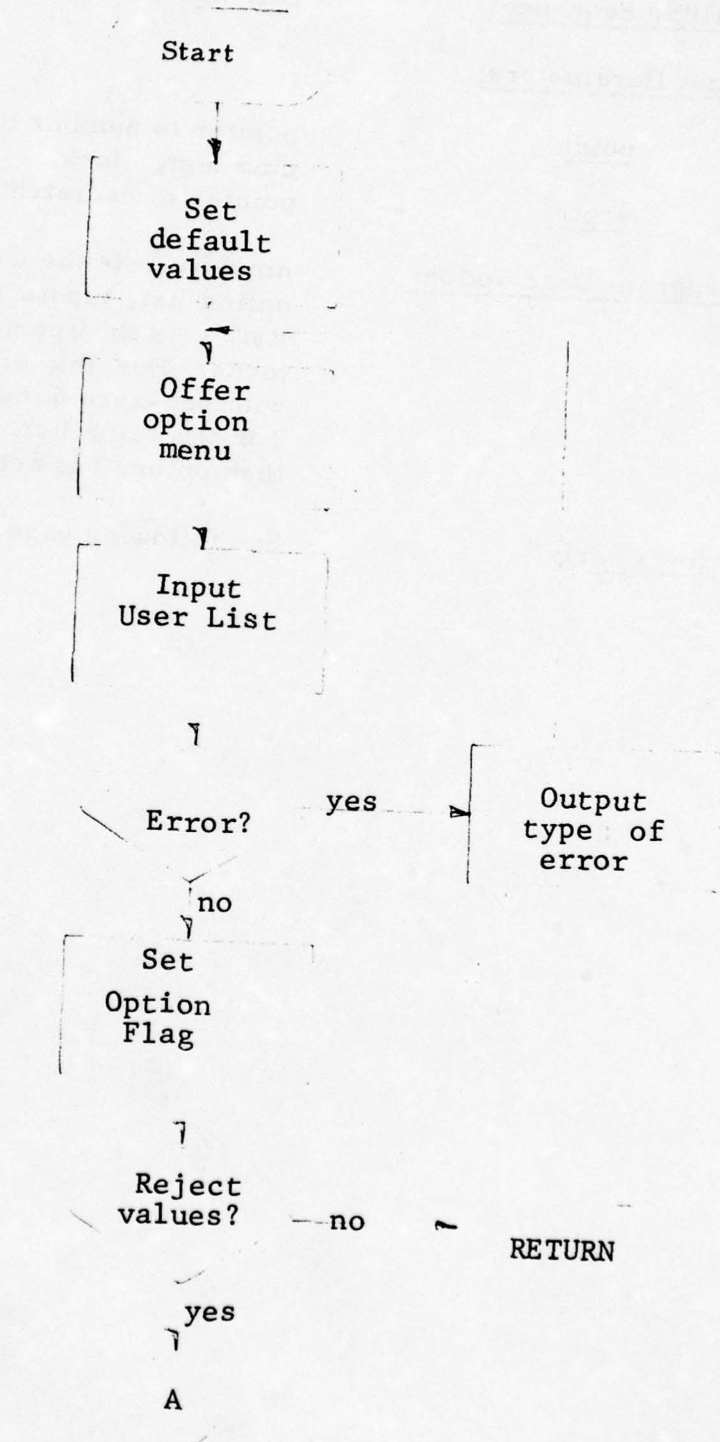
Input Parameters:

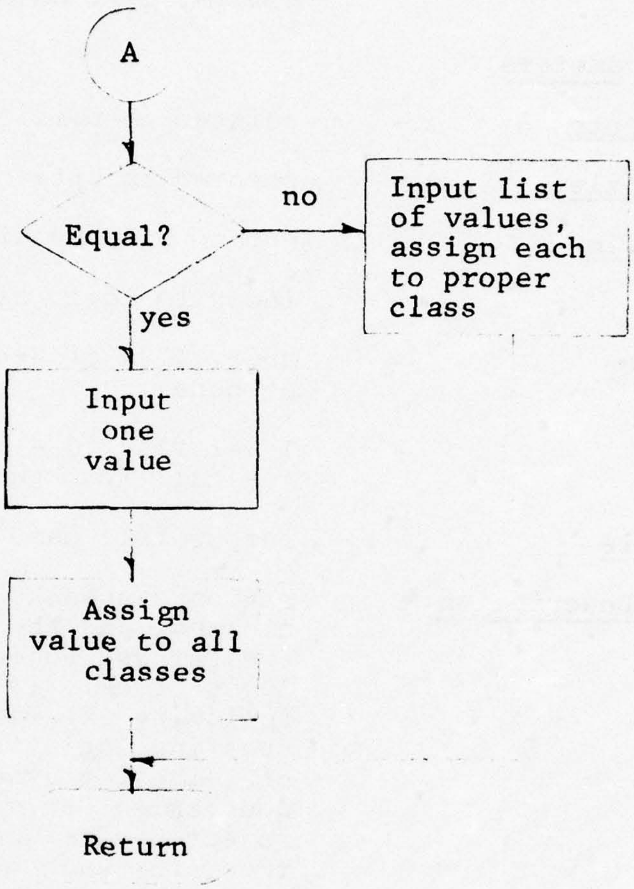
point	-	pointer to number of classes (first word) in nmv logic block.
scptr	-	pointer to "scratch"

Program Description: nmv$fsu sets the default values, offers the option list, inputs and checks the user input list, sets the appropriate flags, and then returns. Possible errors are in entering a 0 with non-zero numbers and entering a 1 and 2 in the same list. If option 4 is selected, then option 3 is automatically selected.

Flow Chart: See following page.

nmv$fsu


Start
Set default values
Offer option menu
Input User List
Error?
yes
Output type of error
no
Set Option Flag
Reject values?
no
RETURN
yes
A


A
Equal?
no
Input list
of values,
assign each
to proper
class
yes
Input
one
value
Assign
value to all
classes
Return

Internal Subroutine Name: nmvlogic

Calling Sequence: call nmvlogic (fileptr, optfile, numdim, dex, nodes, fl, ifile)

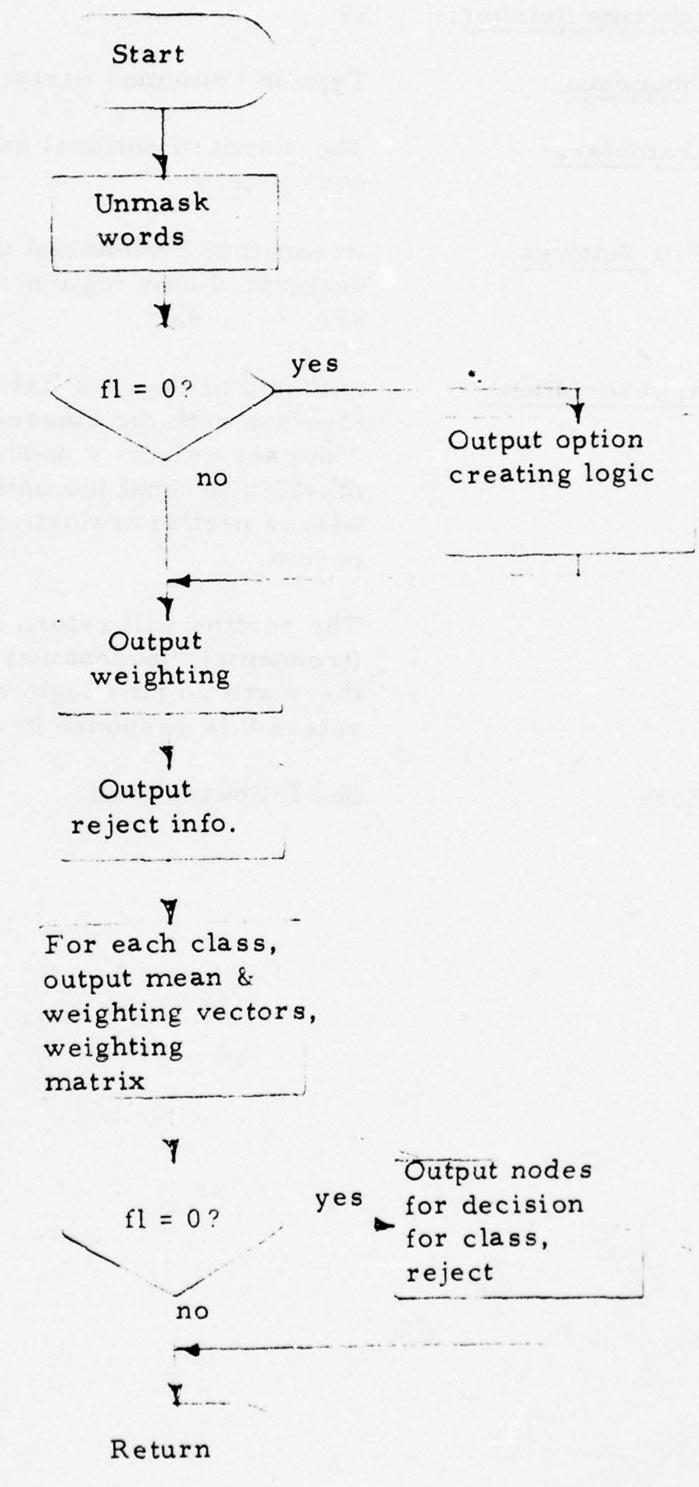
Input Parameters:

- fileptr - pointer to mooslogic file
- optfile - pointer to option_file
- numdim - number of dimensions
- dex - index to logic block
- nodes - array (72) of 4-character node names at node
- fl - level flag (0 - call from listlogic, 1 - call from pwfisher)
- ifile - output file name

Program Description: nmvlogic unmasks the number of classes, and the logic information (weight and reject flags) for a nmv logic block. It then outputs the following information: the option creating logic (if fl = 0), the type of weighting used, the reject boundaries for each class (or no reject used if reject flag = 0), and then, for each class, the mean vector, the weighting vector, and the weighting matrix. If fl = 0, then the routine outputs the logic node to go to, depending on the decision made at the node, before returning.

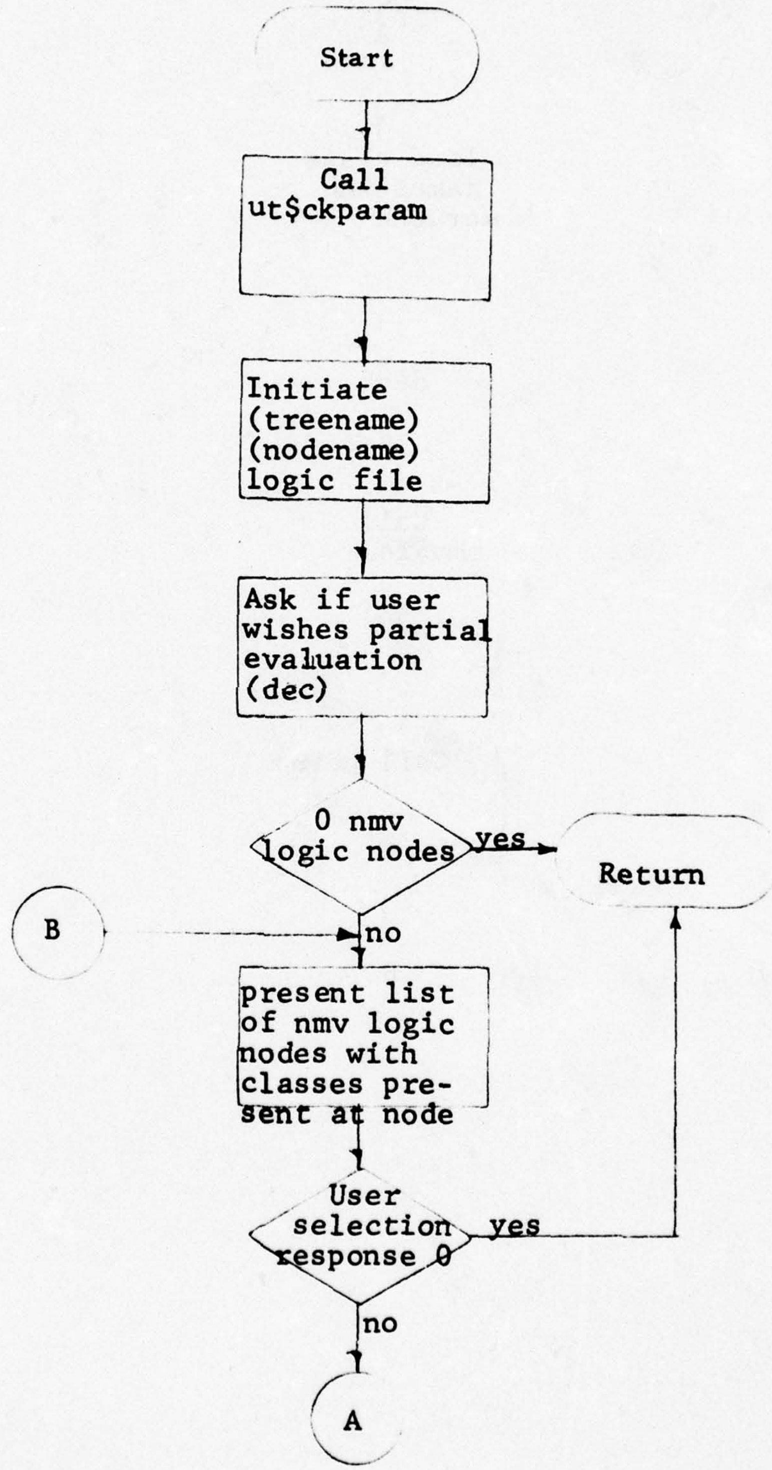
Flow Chart: See following page

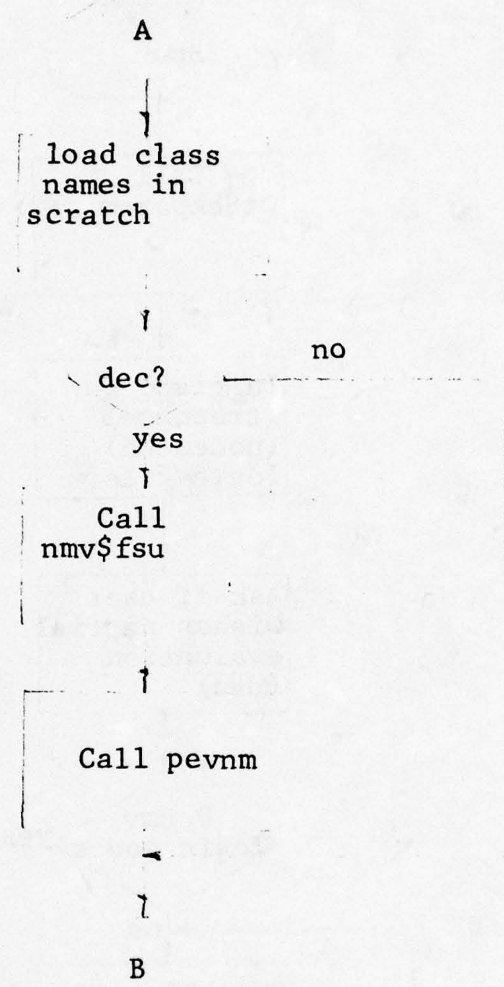
nmvlogic


Start
Unmask words
fl = 0?
yes
no
Output option creating logic
Output weighting
Output reject info.
For each class, output mean & weighting vectors, weighting matrix
fl = 0?
yes
Output nodes for decision for class, reject
no
Return

MOOS Function Name:	nmvmod
MOOS Function Number:	59
Calling Sequence:	Type in "nmvmod [(treename) (nodename)] "
Input Parameters:	The standard optional data set selection parameters.
Output File Settings:	(treename) (nodename) mooslogic file: user designated nmv logic nodes evaluation flags are changed.
Program Description:	nmvmod presents a list of all nmv logic nodes, together with the classes present at the node. The user selects a node, and the routine calls nmv$fsu to input the options and, if the user wishes partial evaluation, call ctsm and pevnm. The routine will return if the logic file for (treename), (nodename) does not exist, if there are no nmv logic nodes, or if the user enters 0 in response to selecting a logic node.
Flow Chart:	See following page.

nmvmod


Start
Call ut$ckparam
Initiate (treename) (nodename) logic file
Ask if user wishes partial evaluation (dec)
0 nmv logic nodes
yes
Return
B
no
present list of nmv logic nodes with classes present at node
User selection response 0
yes
no
A


A
load class names in scratch
dec?
no
yes
Call nmv$fsu
Call pevnm
B

Internal Subroutine Name: nmvprogram

Calling Sequence: call nmvprogram (ii, logicptr)

Input Parameters:

ii — logic node number with nearest mean vector logic (fixed (35))

logicptr — pointer to MOOS logic file (ptr)

Program Description: nmvprogram is the subroutine in the "fortlogc" option of MOOS that generates FORTRAN code for a nearest mean vector logic node

See the subroutine's program listing for a more detailed description of the operation of this subroutine

MOOS Function Name: normxfrm

MOOS Function Number: 129

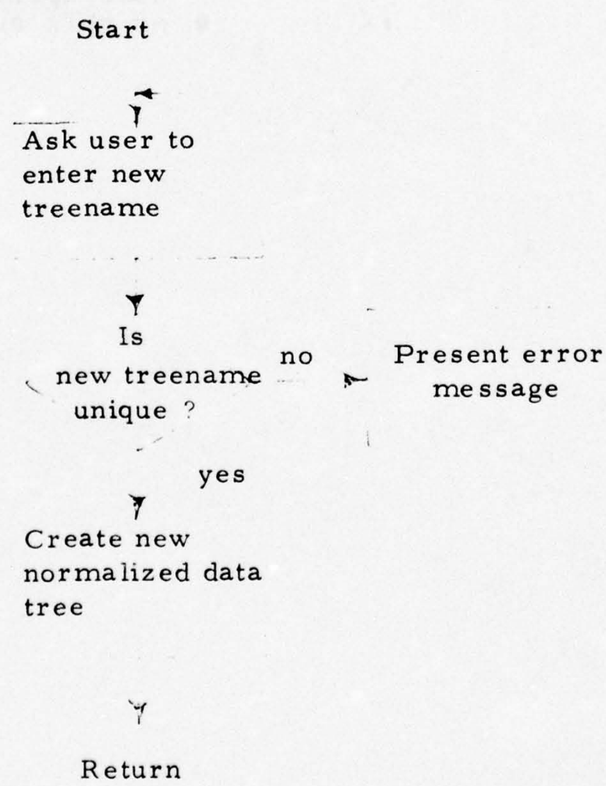
Calling Sequence: type in "normxfrm [(treename)] [nodename)]"

Input Parameters: standard optional data set selection parameters

Program Description: normxfrm asks the user to enter a new tree name. The new tree which is created consists of the original data set. normalized by the standard deviation of each measurement, i.e. each measurement in the normalized data set has unit variance.

Flow Chart:

normxfrm



Internal Subroutine Name: npcos

Calling Sequence: call npcos

Output File Settings: npcos is called by the one-space structure analysis and logic design routines. It creates and sets up the "csdata" file in the proper format (section 3.1) and creates and sets up the "microbuff" file for use in "micro" displays. After projecting the data, it is deposited in the "display" file, starting with word 2.

Program Description: npcos can put up two types of displays; the "macro" view is an overall look at all classes or a selected subset of classes at once. In the view, the user can have a feel for how his data is distributed along a projection vector. The "micro" view is a closer, more detailed look where one, two, or perhaps more classes are displayed. The calling algorithm loads in temporary symbol, tree character, dimensionality, number of classes, macro/micro flag, probability / count flag, and classnames. npcos first checks C7, data projection flag, and projects the data and stores the result in "display" file. Next, it bins the data and stores the binned data, according to class, in the "csdata" file.

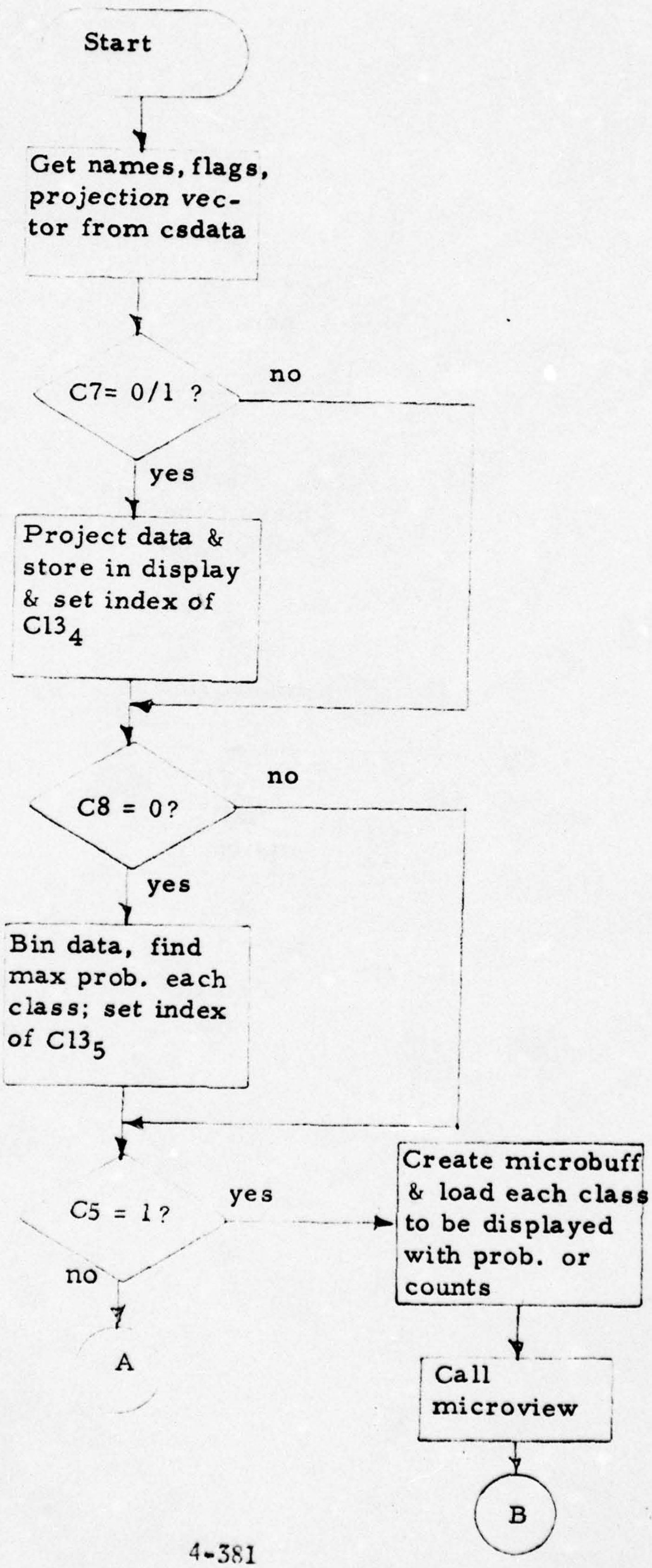
Next, it checks the macro/micro display flag (C5), and if = 1, then it is a micro display. It will then create the "microbuff" file and load the appropriate classes into the file. It then calls the subroutine microview.

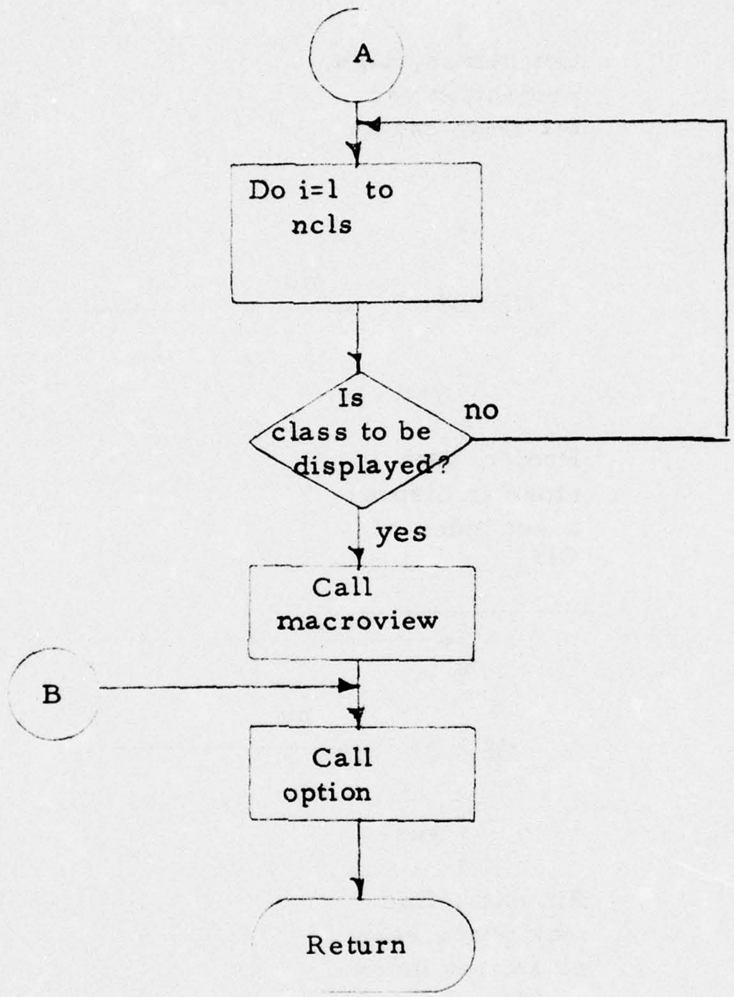
If C5 = 0 then it is a macro view and calls macroview for each displayed class.

npcos then displays scaling, information, number of bins, and ranges of data on the console. It exits by calling option unless the current moos function is pairmod, then it simply returns.

<u>Flow Chart</u>: See following page

npcos


Start
Get names, flags, projection vector from csdata
C7= 0/1 ?
no
yes
Project data & store in display & set index of $C13_4$
C8 = 0?
no
yes
Bin data, find max prob. each class; set index of $C13_5$
C5 = 1?
yes
no
A
Create microbuff & load each class to be displayed with prob. or counts
Call microview
B


A
Do i=1 to ncls
Is class to be displayed?
no
yes
Call macroview
B
Call option
Return

Internal Subroutine Name: onespace

Calling Sequence: call onespace (logicptr, nodeptr)

Input Parameters:

logicptr pointer to MOOS logic file (ptr)

nodeptr pointer to one-space logic node (ptr)

Program Description: onespace is the subroutine in the "fortlogc" option of MOOS which creates FORTRAN code for a one-space group logic node.

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

<u>Internal Subroutine Name</u>:	optdisc
<u>Calling Sequence</u>:	call optdisc (logicptr, pairptr, reject)
<u>Input Parameters</u>:	
<u>logicptr</u>	pointer to MOOS logic file (ptr)
<u>pairptr</u>	pointer to the logic for this pair of classes (ptr)
<u>reject</u>	reject logic node number (fixed (35))
<u>Program Description</u>:	optdisc is the executive for generating FORTRAN code for an optimal discriminant plane logic pair under the "fortlogc" option of MOOS See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: option

Calling Sequence: call option ("opt1", "opt2", ... "optn")

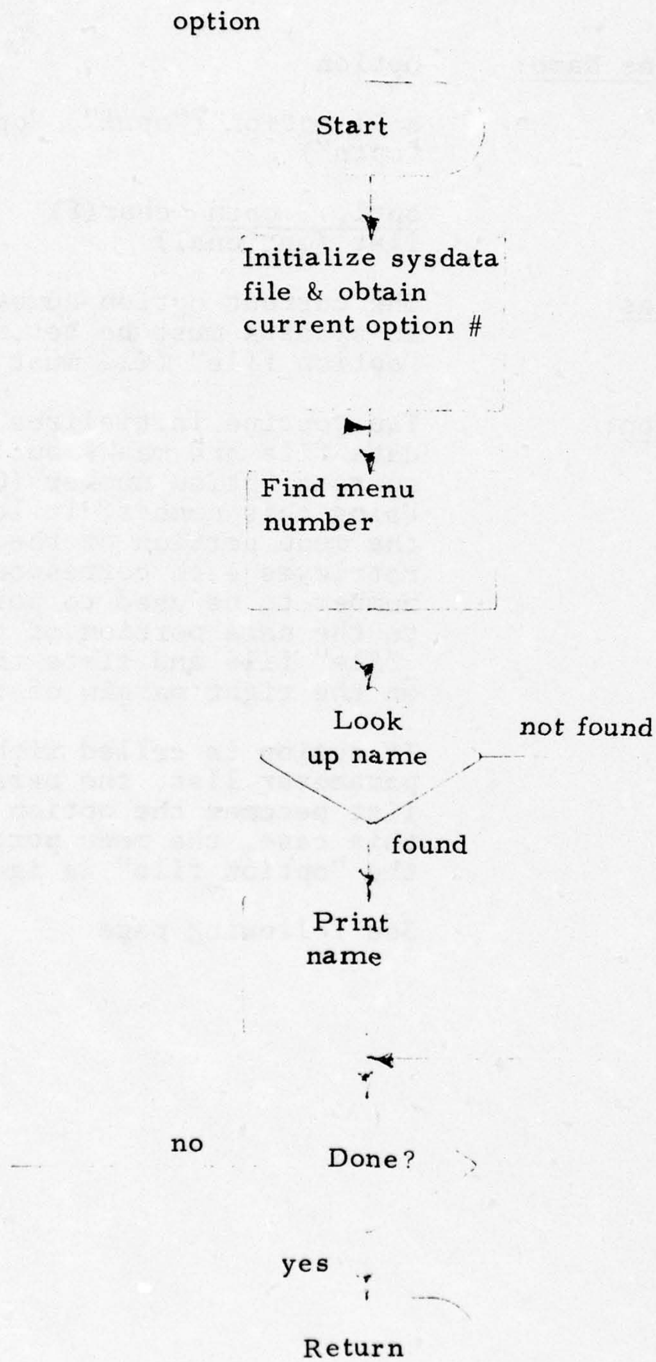
Input Parameters: opt1,...optn char(8) option list (optional)

Input File Settings: The current option number (CSS5) in sysdata must be set and the "option_file" file must be built.

Program Description: The routine initializes the sysdata file and masks out the current option number (CSS5). Using this number, it looks in the menu portion of the file and retrieves each corresponding number to be used to point back to the name portion of the "option_file" file and lists these names on the right margin of the screen.

If option is called with a parameter list, the parameter list becomes the option list. In this case, the menu portion of the "option_file" is ignored.

Flow Chart: See following page


option
Start
Initialize sysdata file & obtain current option #
Find menu number
Look up name
not found
found
Print name
no
Done?
yes
Return

Programmer Aid Name: option$delete

Calling Sequence: Type in "option$delete"

Output File Settings: The specified names will be removed from file "option_file".

Program Description: The "option_file" is initiated and an option name is requested from the keyboard. The name is searched for in the first 511 slots and if found is deleted. Then the menu section of the file for this slot is zeroed out (if the slot number is 256 or less) and all references to the deleted slot are also zeroed in other menus.

Flow Chart: See following page

option$delete

Start

Initiate
option_file

Get name from
keyboard

Look up
program
name

Found? no → Return

yes

Clear out
name

Clear out
menu

Clear all
menu reference

Programmer Aid Name: option$insert

Calling Sequence: Type in "option$insert"

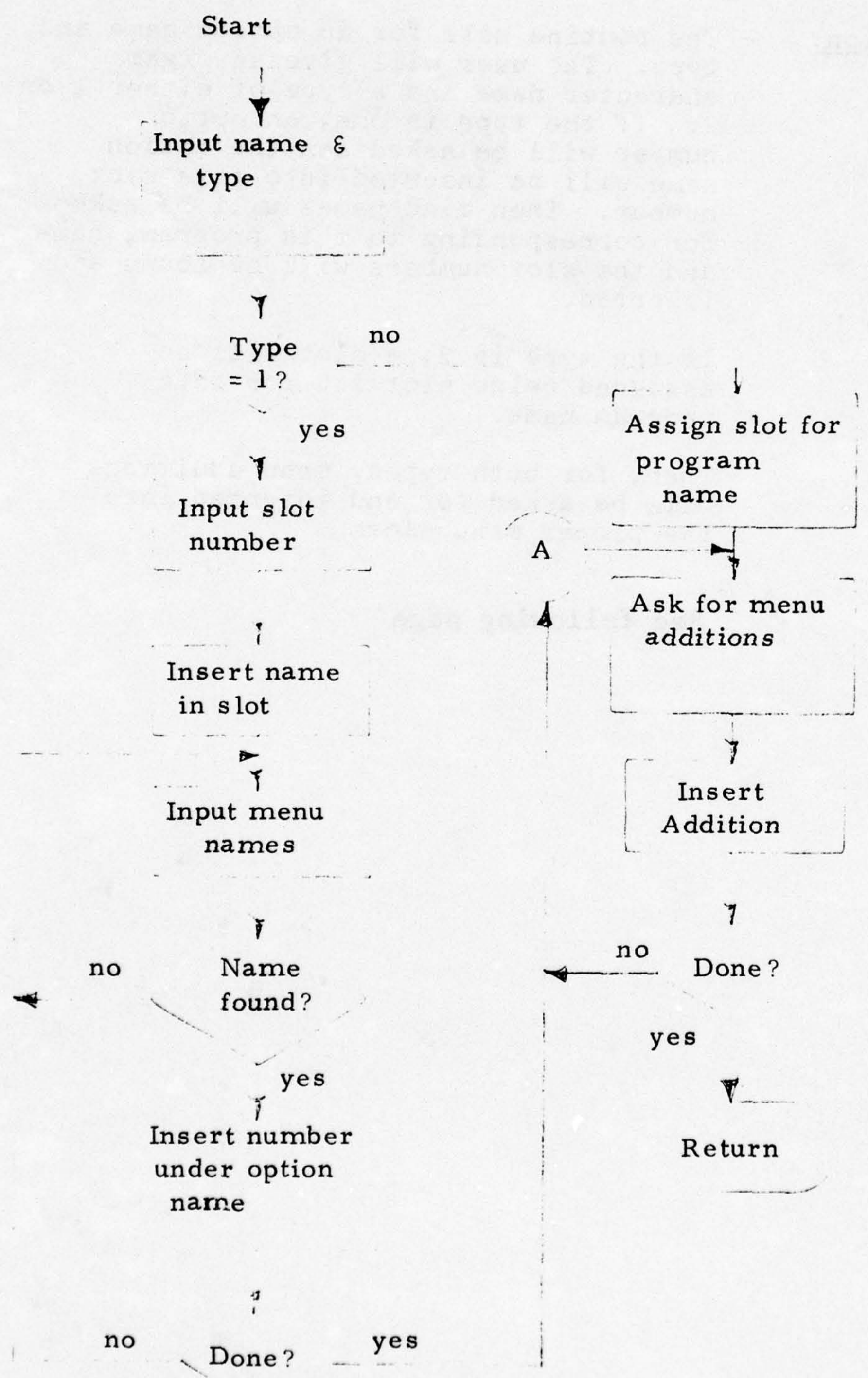
Program Description: The routine asks for an option name and type. The user will give an eight character name and a type of either 1 or 2. If the type is one, an option number will be asked and the option name will be inserted into this slot number. Then menu names will be asked for corresponding to this program, name and the slot numbers will be found and inserted.

If the type is 2, a slot will be assigned below slot 256 for this program name.

Then, for both types, menu additions will be asked for and inserted into the proper menu slots.

Flow Chart: **See following page**

option$insert

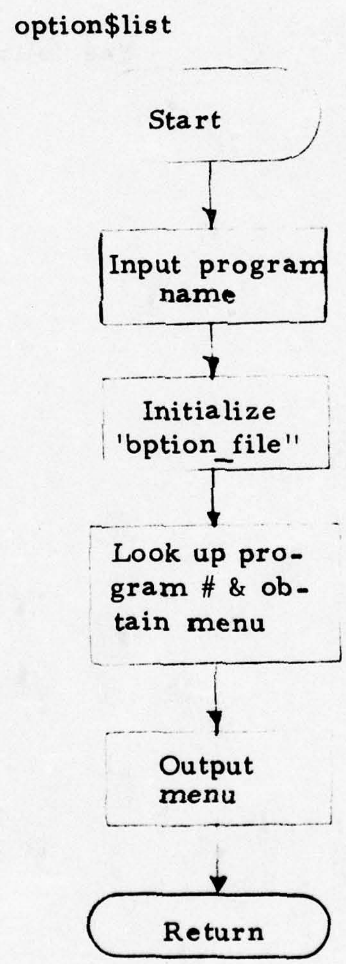


Programmer Aid Name: option$list

Calling Sequence: Type in "option$list"

Program Description: The routine asks for a program name and initiates the "option_file" file. The menu for the name is then typed on the console.

Flow Chart:



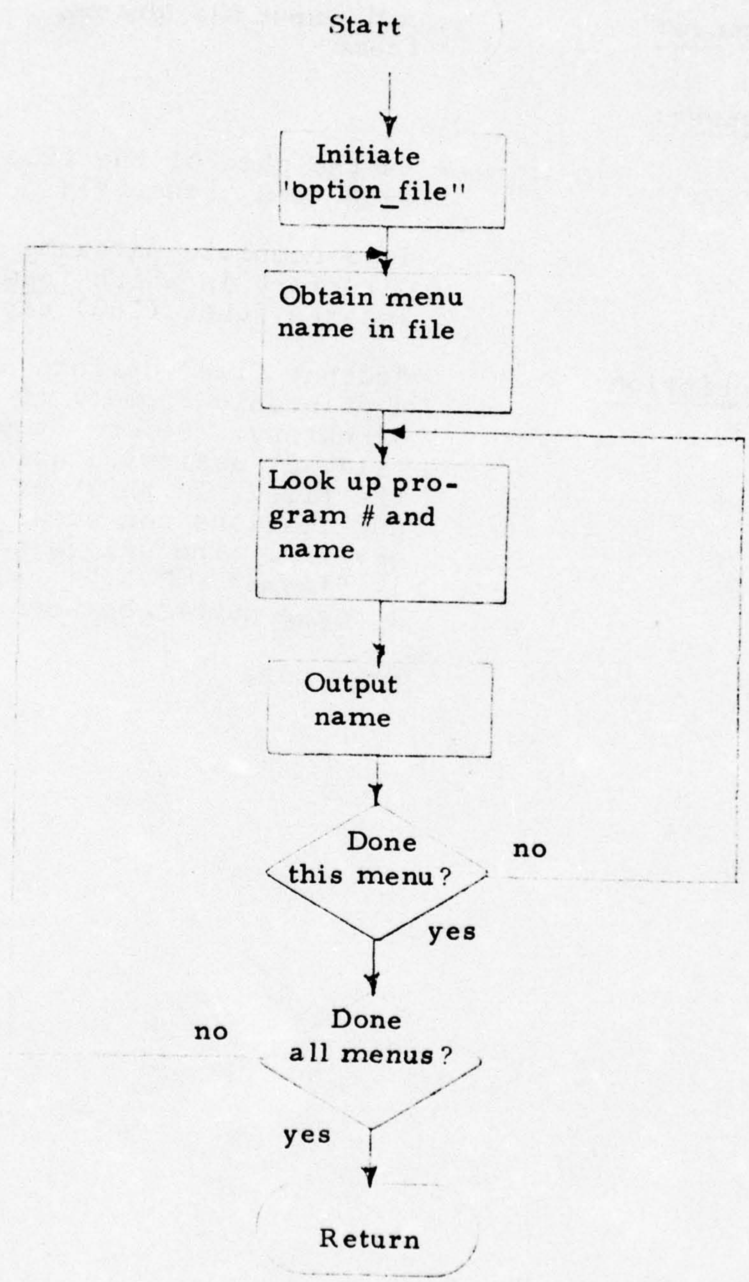
<u>Programmer Aid Name:</u> option$listall

<u>Calling Sequence:</u> Type in "option$listall"

<u>Program Description:</u> The routine initiates the "option_file" file and outputs the menus for all programs listed on the console.

<u>Flow Chart:</u> See following page

option$listall



Internal Subroutine Name: output_file

Calling Sequence: call output_file (dname, fname)

Input Parameters:

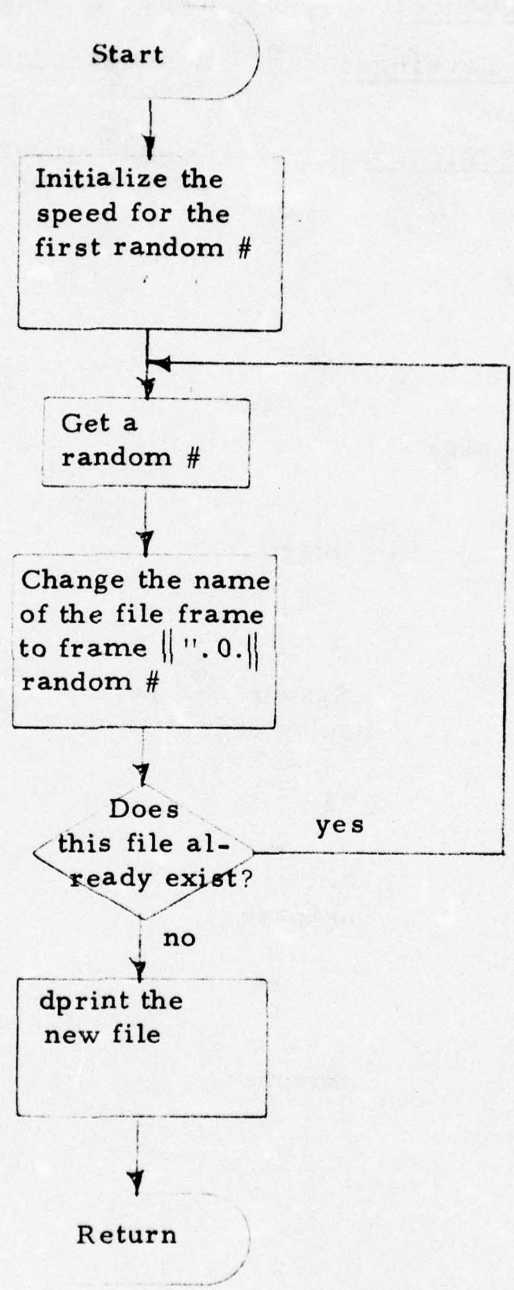
fname — is the name of the file to be outputted [char(*)]

dname — is a complete pathname to the directory in which fname is located [char (168) varying]

Program Description: "output_file" dprints a file(in a printable format) from a given directory. Before it prints the file, it assigns a unique name to the file. In this way no name duplications can occur on outputting. The unique name assigned is "fname" || ".0." || (a random integer number between 1 and 1000)

Flow Chart: Next page

output_file


Start
Initialize the speed for the first random #
Get a random #
Change the name of the file frame to frame ||".0.|| random #
Does this file already exist?
yes
no
dprint the new file
Return

Utility Function Name: page

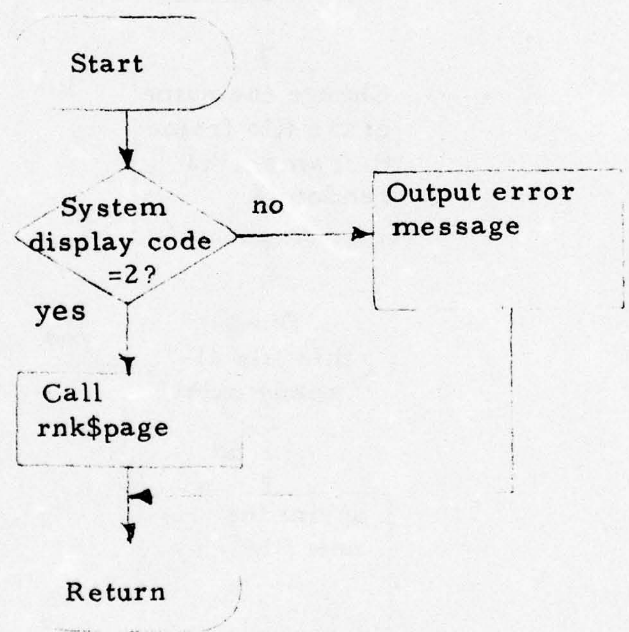
Calling Sequence: Type in "page"

Input File Settings: assumes rank order display file format

Program Description: "page" gives the user the ability to page through a ranking display

Flow Chart:

page



MOOS Function Name: pairmod

MOOS Program Number: 97

Function Call: pairmod [treename] [nodename]

Input Parameters: [treename] and [nodename] specify a data set other than the current data set.

Output File Settings: The logic file is updated to reflect the change in logic for any or all pairs of classes of a fisher node. In addition, a temporary file, "lbuff" is created in the process directory and contains the current modified logic.

Program Description: Initially, the MOOS subroutine "ut$ckparam" is called to determine what the current data set will be. Then "sln$cp" is called to verify that a logic file with at least one completed pair-wise logic node exists for this data set. An error message is printed and control is returned to the command level if there is not.

The user is then asked if any logic nodes are to be combined (the resulting pairwise logic would be a "group" logic). If the user answers yes, mod10 is called.

pairmod then asks the user to enter the desired class pair and checks that this is a valid input. The two node names are stored in words 74 and 75 of the "scratch" file for use by the one and two-space display routine.

The subroutine "pm_list" is called to present the list of logic options to the user and based upon his selection, one of the following subroutines is called:

"mod1" - change the number of fisher thresholds

"mod2" - change the location of a fisher or an arbitrary one-space thresholds

"mod3" - change the number of measurements used in calculating the fisher direction

"mod5" - change the current logic back to "Fisher"

"mod6" - change the current logic to arbitrary one-space

"mod7" - change the current logic to optimal discriminant plane

"mod8" - change the current logic to arbitrary two-space

"mod9" - change the current logic to boolean

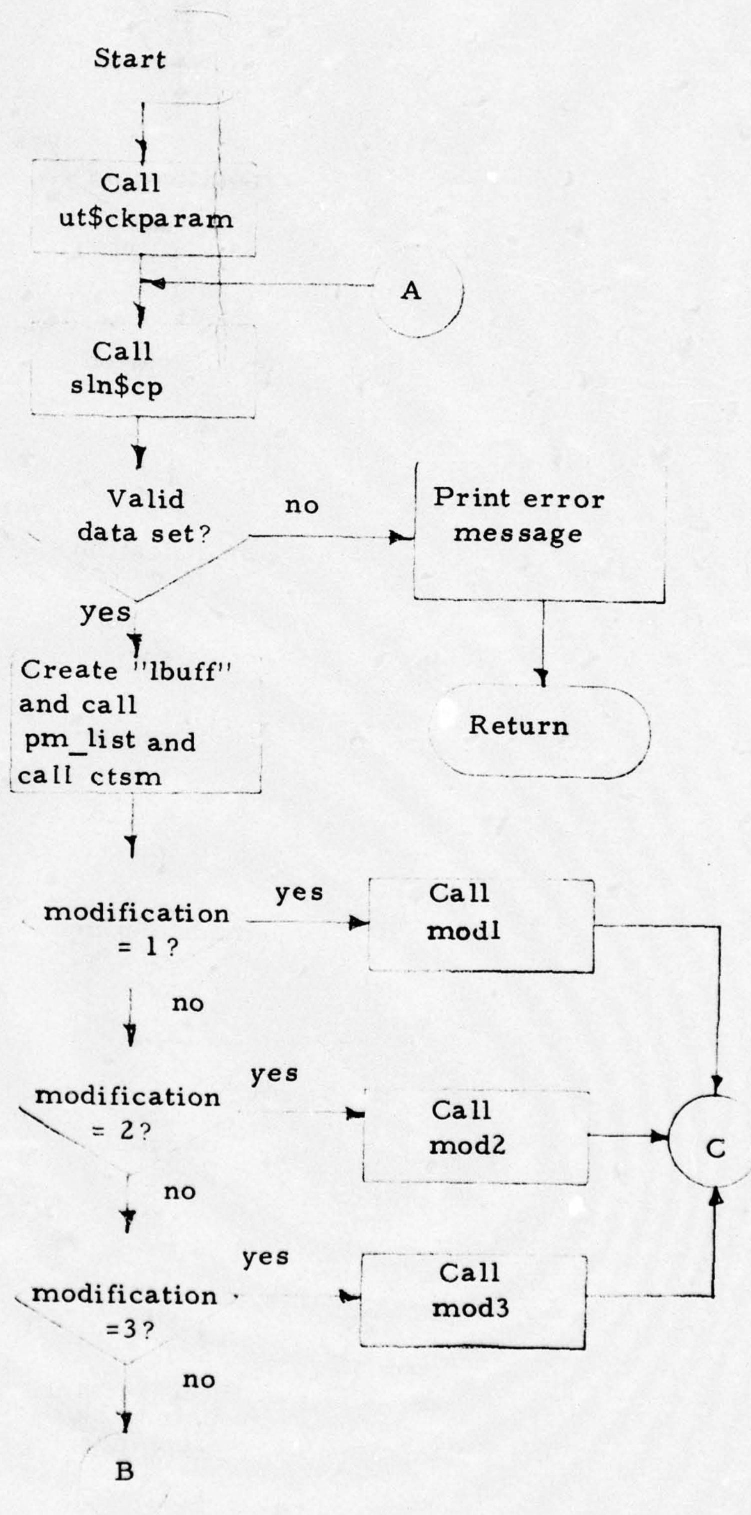
The logic block format for the fisher node is updated to include the auxiliary criteria block if the logic modification is arbitrary one-space, arbitrary two-space, optimal discriminant plane or boolean. Otherwise, upon acceptance of the modified fisher logic, the criteria block for the selected pair is altered.

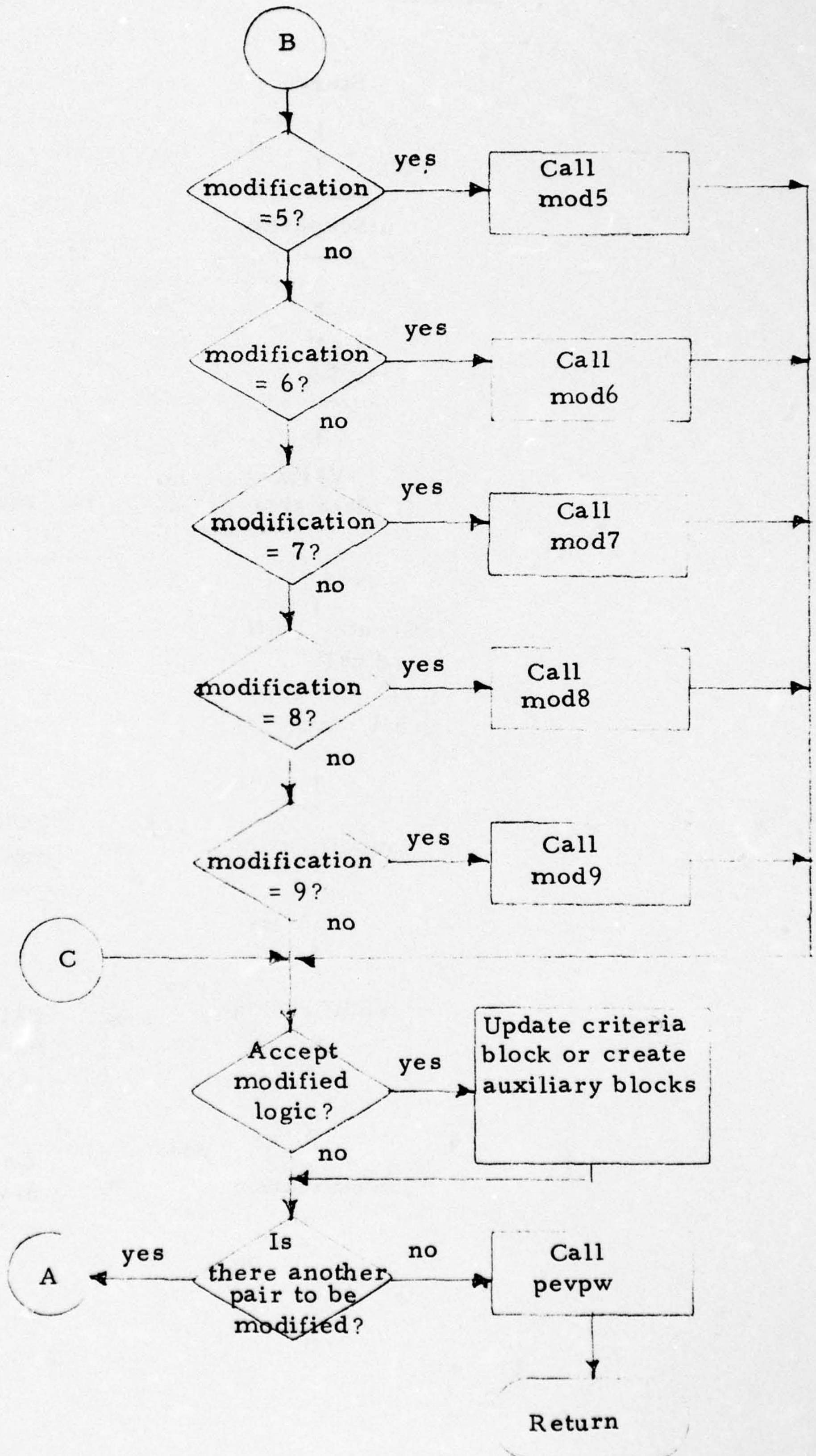
The subroutine "partial2" is used to generate a "mini" confusion matrix that includes the selected pair. The user is asked to accept or reject the new logic based upon the information contained in this confusion matrix.

pairmod exits by calling the moos subroutine "pevpw" which displays a confusion matrix for the classes present at the current node.

Flow Chart: See following page

pairmod


Start
Call ut$ckparam
A
Call sln$cp
Valid data set?
no
Print error message
Return
yes
Create "lbuff" and call pm_list and call ctsm
modification = 1?
yes
Call mod1
no
modification = 2?
yes
Call mod2
C
no
modification =3?
yes
Call mod3
no
B


B
modification =5?
yes
Call mod5
no
modification = 6?
yes
Call mod6
no
modification = 7?
yes
Call mod7
no
modification = 8?
yes
Call mod8
no
modification = 9?
yes
Call mod9
no
C
Accept modified logic?
yes
Update criteria block or create auxiliary blocks
no
Is there another pair to be modified?
yes
A
no
Call pevpw
Return

Internal Subroutine Name: pairprogram

Calling Sequence: call pairprogram (i, logicptr, nlnodes)

Input Parameters:

i — logic node number with pairwise logic (fixed (35))

logicptr — pointer to MOOS logic file (ptr)

nlnodes — number of logic nodes in this logic file

Program Description: pairprogram is the executive routine for generating FORTRAN code for a pairwise logic node under the "fortlogc" option of MOOS

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: pairwise_logic

Calling Sequence: call pairwise_logic (lptr, aptr, nptr, dcptr, erptr, ndim, cn, nnum, ln, max, cflag)

Input Parameters:

lptr	ptr	pointer to mooslogic file
aptr	ptr	pointer to apriori probability section of mooslogic file
nptr	ptr	pointer to current logic node in mooslogic file
dcptr	ptr	pointer to the vector being evaluated
erptr	ptr	pointer to the next available entry in the pair_error_file
ndim		fixed (35) no. of dimensions
cn		fixed (35) current logic node number
nnum		128 fixed (35) array of indices referring to the table of class names in the mooslogic file. If i is a logic node number, nnum(i) = index to the class name associated with i
ln		(128, 72) fixed (35) If i is a pairwise logic node number, ln(i, 1) = the first logic node number beneath i, ln(i,2) = the next logic node number, etc.
cflag		fixed (35) Usually set to 0. If cflag is set to 1, pairwise_logic outputs vote counts for each classified vector to the user output stream.

Output Parameters:

max	fixed(35) The vote count at which the vector was classified. This parameter is used by partial pairwise evaluation (pevpw).
cn	fixed(35) The logic node number to which the vector is assigned.

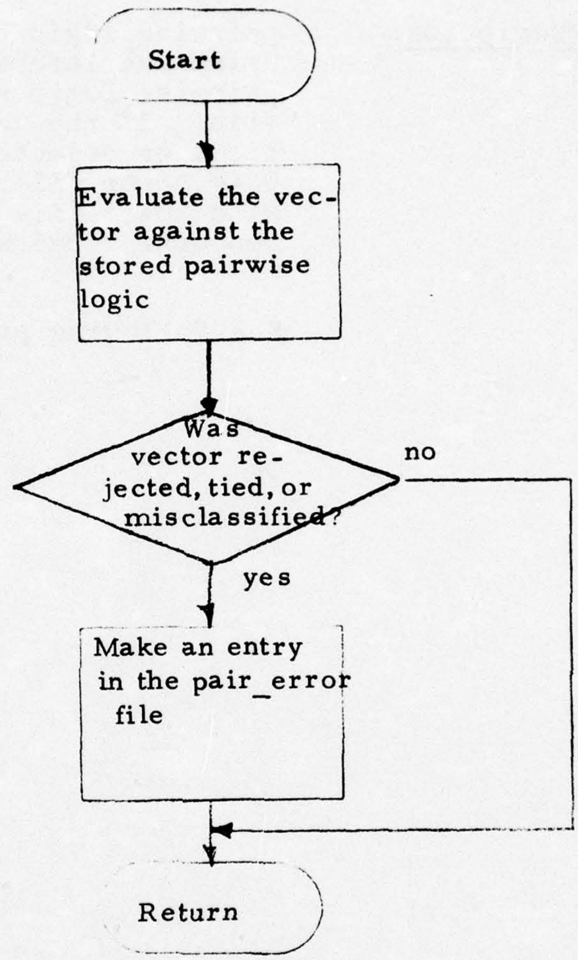
Input File Settings: The word pointed to by "erptr" in the pair_error_file. Must be set to the logic node number of the true class. The next word in this file must be set to the minimum vote count threshold.

Output File Settings: If a vector is incorrectly classified, tied, or rejected, an entry is created in the pair_error_file for that vector.

Program Description: pairwise_logic classifies a given vector using the information stored at a pairwise logic node in a mooslogic file. If the vector is misclassified, tied, or rejected, an entry is made in pair_error_file. See Section 3.1 pairwise_logic_file details for information concerning pairwise logic evaluation.

Flow Chart: See following page

pairwise_logic



Internal Subroutine Name: partial2

Calling Sequence: call partial2 (type, pairptr, lptr, ndim, ptrs, nodeno)

Input Parameters:

type — integer corresponding to modified logic

1 fisher
2 arbitrary two-space
3 arbitrary one-space
4 optimal discriminant plane
5 boolean

other — see "mod1"

Input File Setting: The "lbuff" file must be set in appropriate logic file format

Program Description: This is the evaluation routine called by all of "pairmod"'s logic modification routines. The parameter "type" determines what type of evaluation is to occur, types 2 and 4 are evaluated together, all others separately.

For fisher logic, each vector is projected upon the fisher direction and its location with respect to the thresholds is determined. This data is tabulated for all the vectors.

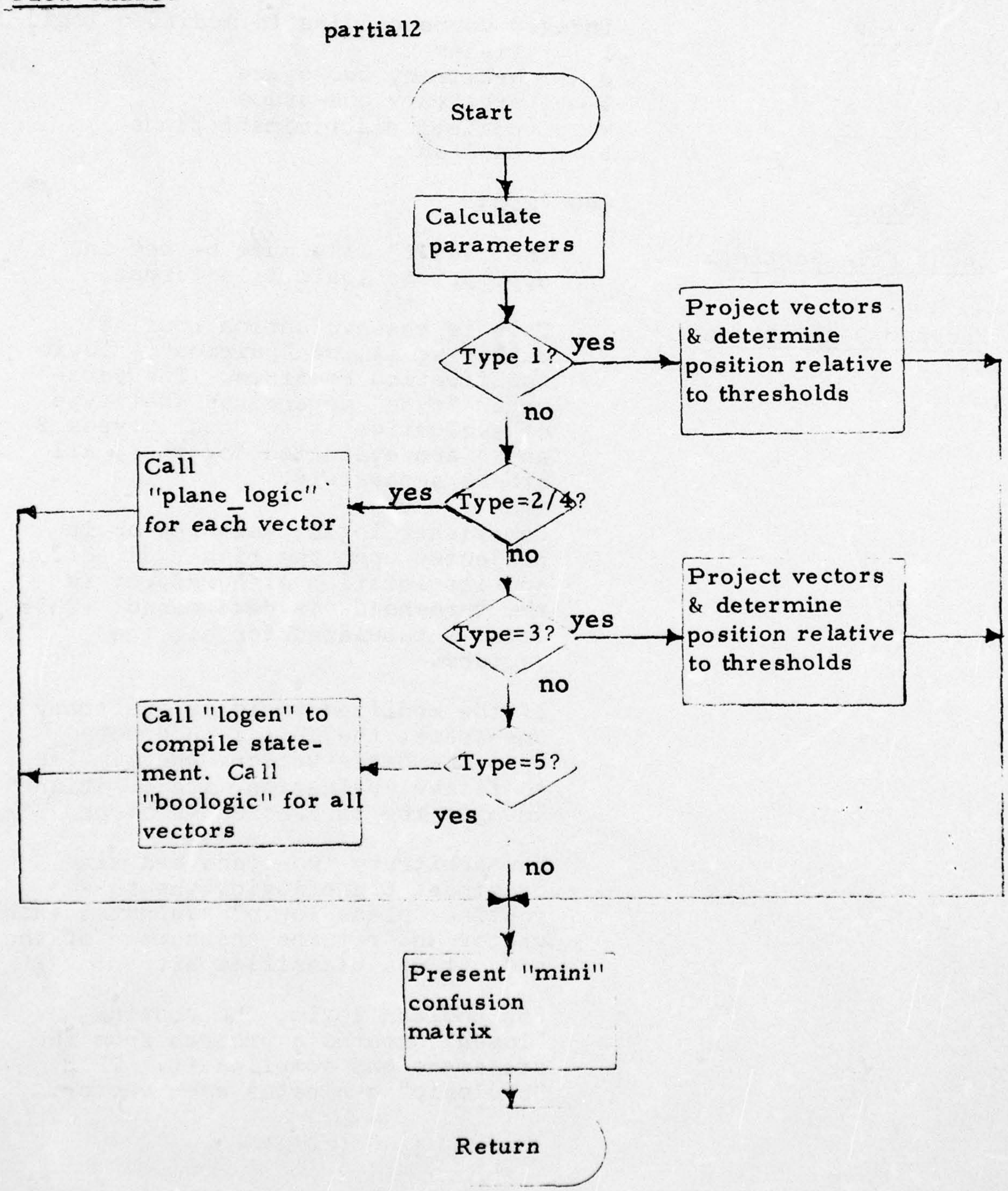
If the modified logic is arbitrary one-space, the data is projected upon the basis vector, and similar to fisher evaluation, its position amongst the thresholds is stored.

For arbitrary two-space and discriminant plane logic, the subroutine "plane_logic" evaluates each vector and returns the number of the node it was classified as

For boolean logic, the routine "logen" creates a program from the statement and compiles it. Then "boologic" evaluates each vector.

The tabulated data is then presented as a "mini" confusion matrix consisting of the two selected classes and control is returned to the calling program.

Flow Chart:



Internal Subroutine Name: pc$add

Calling Sequence: call pc$add

Input Parameters:

char	next sequential character from input program (char (1) external static)
length	number of characters in symbol (fixed (35) external static)

Output Parameter:

symbol	updated next symbol (char (132) external static)

Program Description: pc$add concatenates the current character onto "symbol"

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: pc$gnbc

Calling Sequence: call pc$gnbc

Input Parameters:

progptr — pointer to input FORTRAN program (ptr external static)

iloc — current character position in the input program (fixed (35) external static)

Output Parameters:

char — next non-blank character (char (1) external static)

Program Description: pc$gnbc retrieves the next non-blank character from the input program

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: pc$gnc

Calling Sequence: call pc$gnc

Input Parameters:

progptr — pointer to FORTRAN program (ptr external static)

iloc — current character position in the input program (fixed (35) external static)

Output Parameters:

char — next character (char(1) external static)

Program Description: pc$gnc retrieves the next sequential character from the input program

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name:	pc$gns
Calling Sequence:	call pc$gns
Input Parameters:	
progptr	pointer to input program (ptr external static)
iloc	current character position in the input program (fixed (35) external static)
char	next input character (char(1) external static)
Output Parameters:	
symbol	next symbol (char(132) external static)
Program Description:	pc$gns retrieves the next symbol from the input program See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name:	pc$psym
Calling Sequence:	call pc$psym (nchars, string)
Input Parameters:	
nchars	number of characters to output (fixed (35))
string	character string to output (char (132))
oprogptr	pointer to output FORTRAN program (ptr external static)
Program Description:	pc$psym outputs a string to the output FORTRAN program
	See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: pconversion

Calling Sequence: call pconversion

Input Parameters:

progptr — pointer to unformatted FORTRAN program (ptr external static)

oprogptr — pointer to output FORTRAN program to be created (ptr external static)

Program Description: pconversion converts free-formatted FORTRAN into card image FORTRAN.

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: pcos

Calling Sequence: call pcos

Input File Settings: "pcos" expects the class names in words $D13_{1,1}$, $D13_{2,1}$, $D13_{3,1}$,... of the "csdata" file, the"macro/micro" file set, and the projected data as determined by the moos function :probconf".

Output File Settings: The "csdata" and possibly "microbuff" files are built to reflect the current histogram.

Program Description: This routine is similar to "npcos" except this routine is only called by the utility function "histgram" which is an option of the moos function "probconf". Therefore, "pcos" is the display generation program of "probconf" while "npcos" is the routine for all other one-space functions.

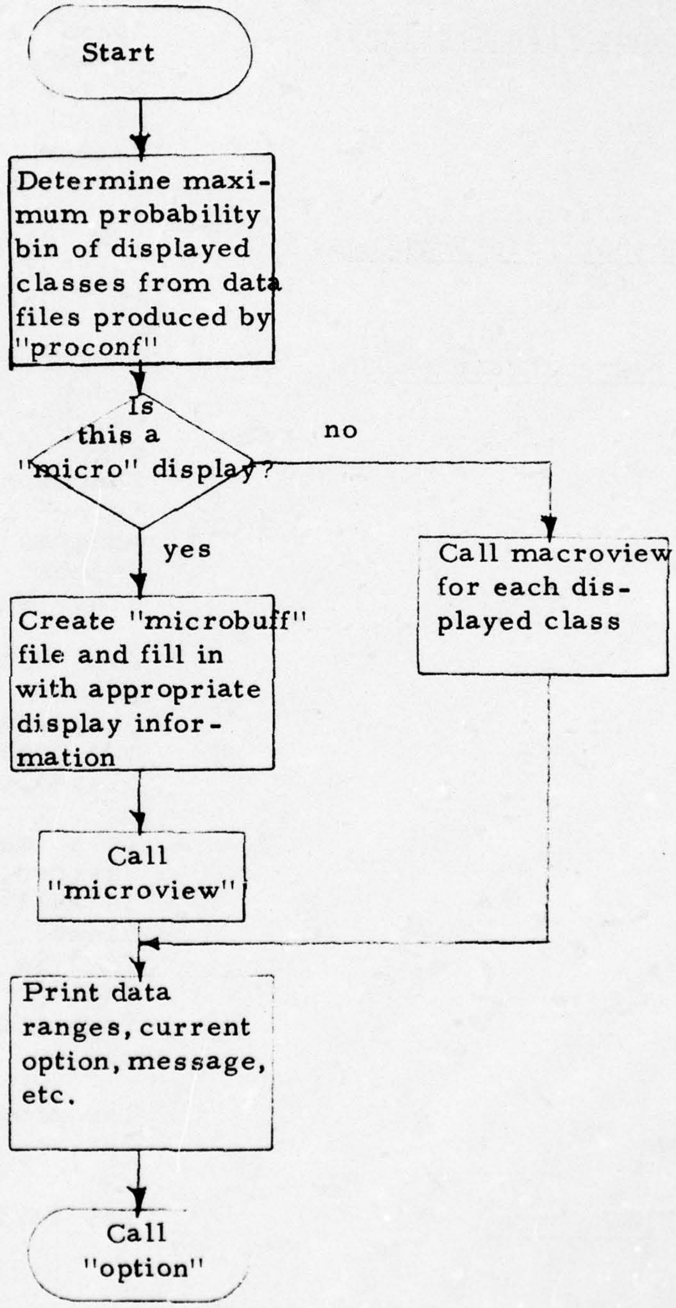
Each data projection file corresponding to the classes to be displayed are examined and the maximum probability bin of all displayed classes is determined.

If a "macro" type display is desired, the subroutine "macroview" is called for each class to be viewed. Otherwise, the "microbuff" file is created and filled in with the appropriate display information (see"npcos") and "microview" is called.

The program exits by calling option.

Flow Chart: See following page

pcos



Start
Determine maximum probability bin of displayed classes from data files produced by "proconf"
Is this a "micro" display?
no
yes
Create "microbuff" file and fill in with appropriate display information
Call macroview for each displayed class
Call "microview"
Print data ranges, current option, message, etc.
Call "option"

AD-A033 437 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
MULTICS OLPARS OPERATING SYSTEM.(U)
SEP 76 D B CONNELL, K N KLINGBAIL F30602-75-C-0226
UNCLASSIFIED PAR-74-25-B RADC-TR-76-271-VOL-2 NL

6 OF 7
AD
A033437

Internal Subroutine Name: pevbx

Calling Sequence: call pevbx (cn, ptrs)

Input Parameters:

cn	fixed (35) logic node at which closed decision boundary logic is being created	
ptrs	(5) ptr	ptrs(1) - sysdata ptrs(2) - scratch ptrs(3) - display ptrs(4) - treename ptrs(5) - mooslogic

Output File Settings: pevbx sets the temporary symbols of the vectors being evaluated to the logic node number to which they are assigned.

The display file is set up in the conmatsm display file format.

Program Description:

pevbx evaluates closed decision boundary logic at a closed decision boundary logic node. The results of the evaluation are presented on the screen in confusion matrix format.

For a more detailed description of the operation of pevbx, see the program listing documentation.

Internal Subroutine Name: pevgl

Calling Sequence: call pevgl(cn,lptr)

Input Parameters:

cn — fixed(35) logic node at which logic is to be evaluated

lptr — ptr pointer to mooslogic file

Output File Settings: pevgl sets temporary symbols associated with vectors being evaluated to the logic node numbers at which vectors are classified.

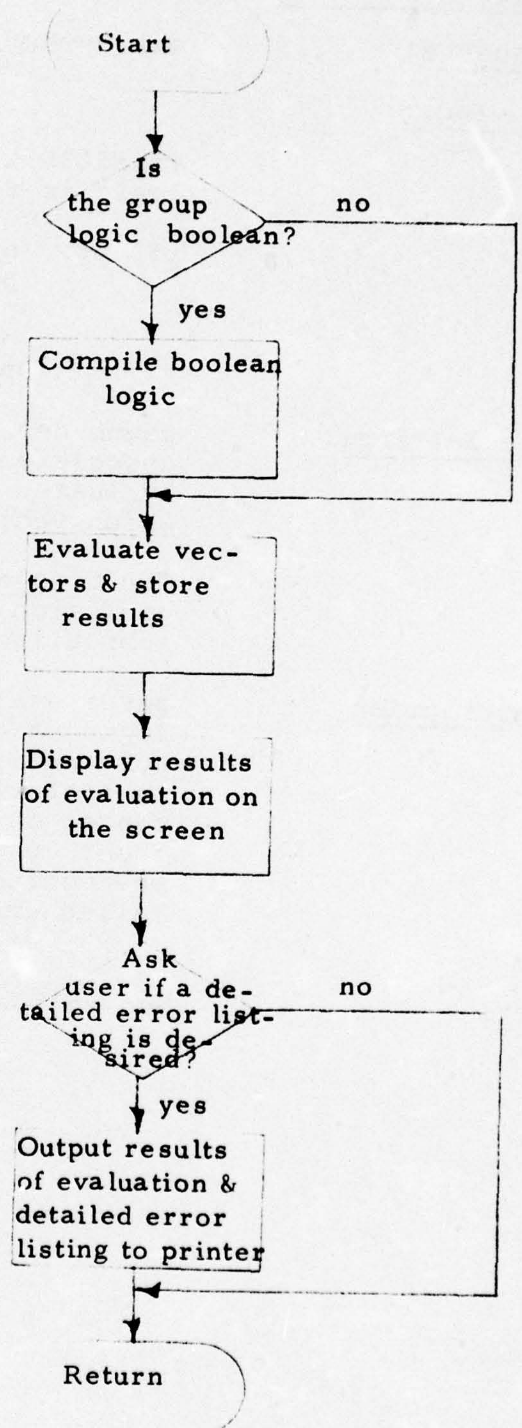
If a hardcopy of results is requested, an output file called eval_file is created where results are stored prior to printout.

Program Description: pevgl evaluates group logic at any group logic node in the mooslogic file. Types of group logic currently implemented are one-space, two-space, and boolean. The results of evaluation are presented on the screen in a confusion matrix format listing assigned logic nodes across the top and true classes on the left-most column.

The user has the option of outputting results on the screen and/or detailed error listings to the printer.

Flow Chart: See following page

pevgl


Start
Is the group logic boolean?
no
yes
Compile boolean logic
Evaluate vectors & store results
Display results of evaluation on the screen
Ask user if a detailed error listing is desired?
no
yes
Output results of evaluation & detailed error listing to printer
Return

Internal Subroutine Name: pevnm

Calling Sequence: call pevnm(cn,ptrs)

Input Parameters:

cn — fixed(35) logic node at which logic is to be evaluated

ptrs — (5) ptr
ptrs(1) - sysdata
ptrs(2) - scratch
ptrs(3) - display
ptrs(4) - treename
ptrs(5) - mooslogic

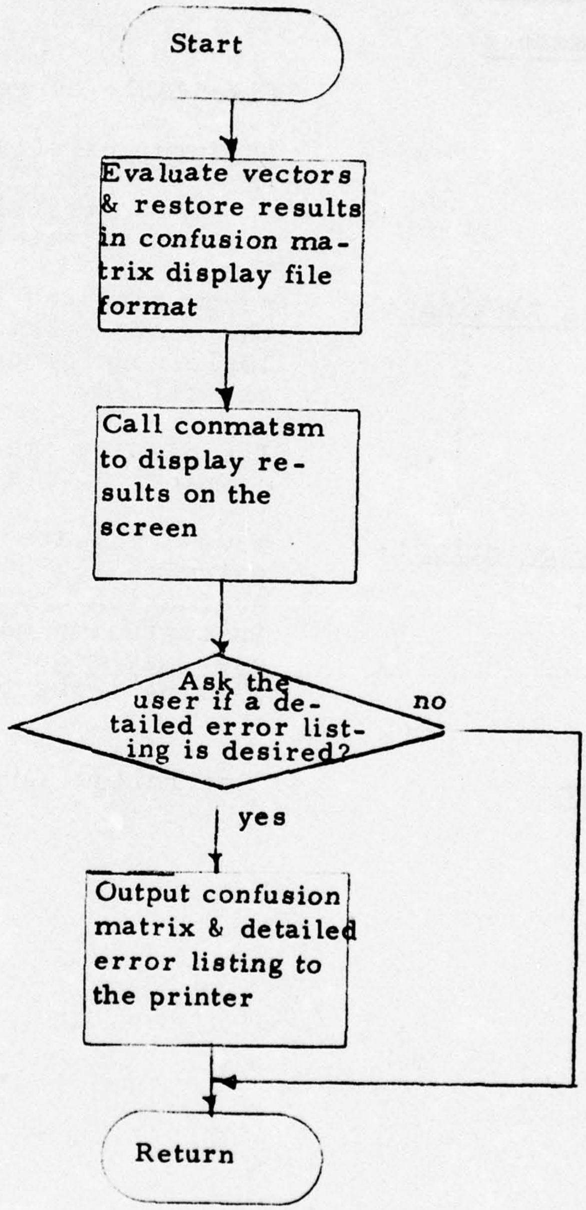
Output File Settings: pevnm sets temporary symbols associated with the vectors being evaluated to logic node numbers at which vectors are classified.

The display file is set up in the confusion matrix format for internal subroutine conmatsm.

Program Description: pevnm evaluates nearest mean vector logic at a nearest mean vector logic node. Evaluation results are stored in confusion matrix form and presented on the screen by conmatsm. The user may then elect to output the confusion matrix and a detailed error listing to the printer.

Flow Chart: See following page

pevnm


Start
Evaluate vectors & restore results in confusion matrix display file format
Call conmatsm to display results on the screen
Ask the user if a detailed error listing is desired?
no
yes
Output confusion matrix & detailed error listing to the printer
Return

Internal Subroutine Name: pevpw

Calling Sequence: call pevpw (cn, ptrs)

Input Parameters:

cn	fixed(35)	current logic node
ptrs	(5) ptr	ptrs(1) - sysdata ptrs(2) - scratch ptrs(3) - display ptrs(4) - treename

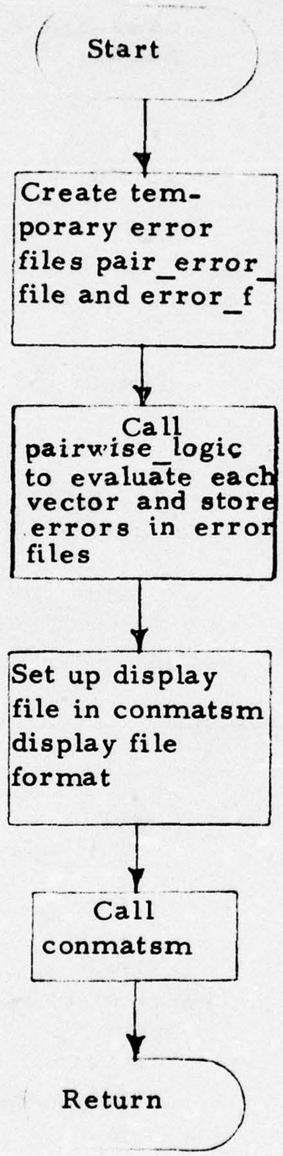
Output File Settings: pevpw sets the temporary symbols of the vectors being evaluated to the logic node numbers to which they are classified.

The display file is set up in the conmatsm display file format.

Program Description: pevpw evaluates pairwise logic at a pairwise logic node. The results of evaluation are presented on the screen in confusion matrix format and the user may request a detailed error listing to be printed.

Flow Chart: See following page

pevpw


Start
Create temporary error files pair_error_file and error_f
Call pairwise_logic to evaluate each vector and store errors in error files
Set up display file in conmatsm display file format
Call conmatsm
Return

Internal Subroutine Name: plane_logic

Calling Sequence: call plane_logic (cn, nptr, dcptr, ndim)

Input Parameters:

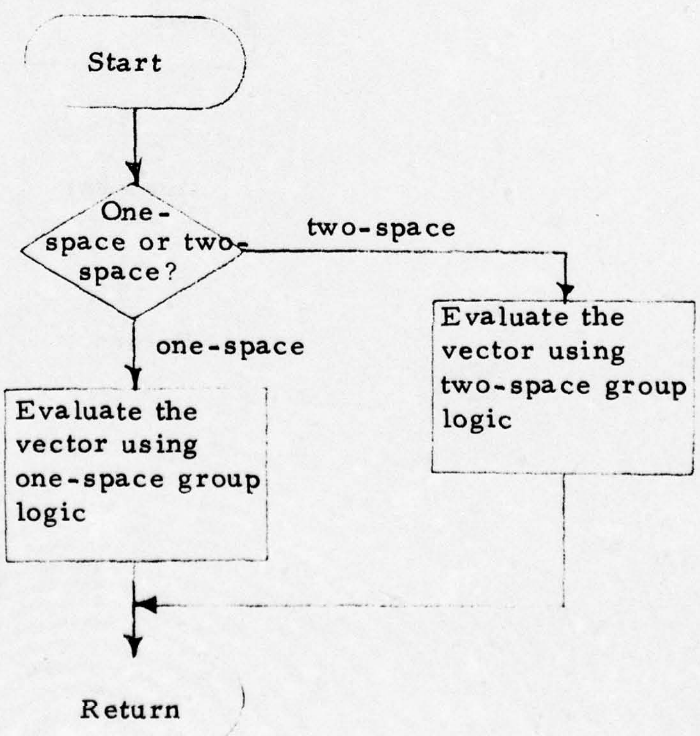
cn	fixed(35)	current logic node
nptr	ptr	pointer to logic block in mooslogic file
dcptr	ptr	pointer to a vector
ndim	fixed(35)	no. of dimensions

Output Parameters:

cn	fixed(35)	logic node to which the vector is assigned

Program Description: plane_logic evaluates a given vector for one-space or two-space, group logic

Flow Chart: plane_logic



Internal Subroutine Name: plinguistic

Calling Sequence: call plinguistic (logicptr, pairptr, reject)

Input Parameters:

- logicptr — pointer to MOOS logic file (ptr)
- pairptr — pointer to the logic for this pair (ptr)
- reject — reject logic node number (fixed (35))

Program Description: plinguistic is the subroutine under the "fortlogc" option of MOOS that generates FORTRAN code for a class pair that uses linguistic pairwise logic.

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: pm_list

Calling Sequence: call pm_list (type, nthrsh, cp, c2)

Input Parameters:

type — The integer corresponding to the current logic

1 - Fisher
2 - optimal discriminant plane
3 - any arbitrary one-space
4 - arbitrary two-space
5 - boolean

nthrsh — number of thresholds used in logic evaluation

cp — the two-character concatenation of the class pair display symbols

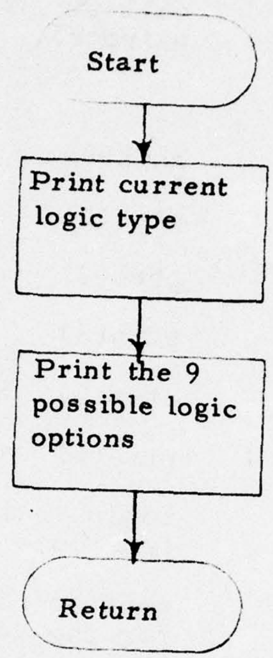
Output Parameters:

c2 — the integer corresponding to the logic modification desired

Program Description: The current logic type is printed and the user is asked to select the modification desired. This input value is converted to a number from 1 to 10, or -1, depending on the selected logic and this value is returned to "pairmod" for use in determining which logic modification subroutine to call.

Flow Chart: See following page

pm_list


Start
Print current logic type
Print the 9 possible logic options
Return

Internal Subroutine Name: pm_list$option

Calling Sequence: call pm_list$option (ptrs, mod_no, pairptr)

Input Parameters:

ptrs — an array of 5 pointers

ptrs(1)	pointer to top of "sysdata" file
ptrs(2)	pointer to word 73 of the "scratch" file
ptrs(3)	pointer to top of "display" file
ptrs(4)	pointer to top of treename file
ptrs(5)	pointer to top of logic file

mod_no — logic modification number, returned from moos subroutine "pm_list"

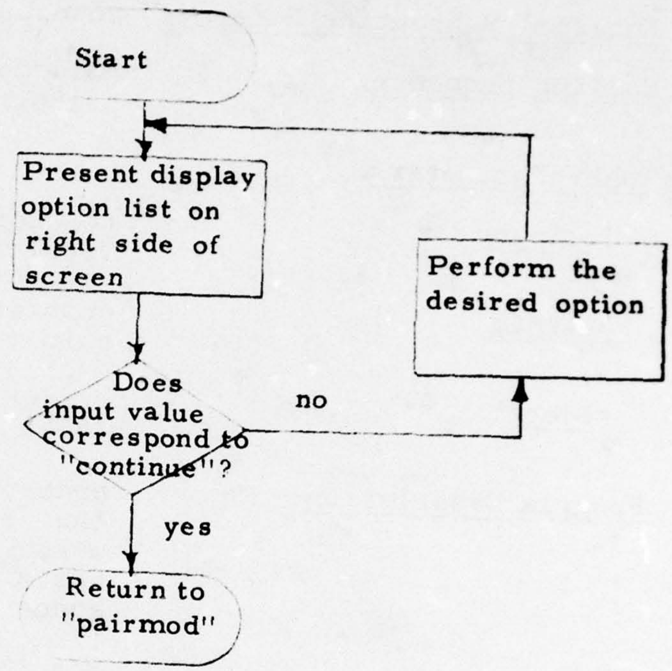
pairptr — pointer into the header information for the desired class pair in the logic block of the selected fisher node

Program Description: This subroutine, depending upon the value of the parameter "mod_no", supplies the display option list present at the upper right-hand corner of the screen during execution of the moos function "pairmod".

After the user inputs the number of the desired display option, this value is interpreted and the appropriate moos utility function is called. The user will remain in "pm_list$option" until the option number for "continue" is entered and control is returned to "pairmod".

Flow Chart: See following page

pm_list$option



Internal Subroutine Name: ponespace

Calling Sequence: call ponespace (logicptr, pairptr, reject)

Input Parameters:

logicptr — pointer to the MOOS logic file (ptr)

pairptr — pointer to the logic for this pair (ptr)

reject — reject logic node number (fixed (35))

Program Description: ponespace is the subroutine in the "fortlogc" option of MOOS which creates FORTRAN code for a class pair that uses one-space logic

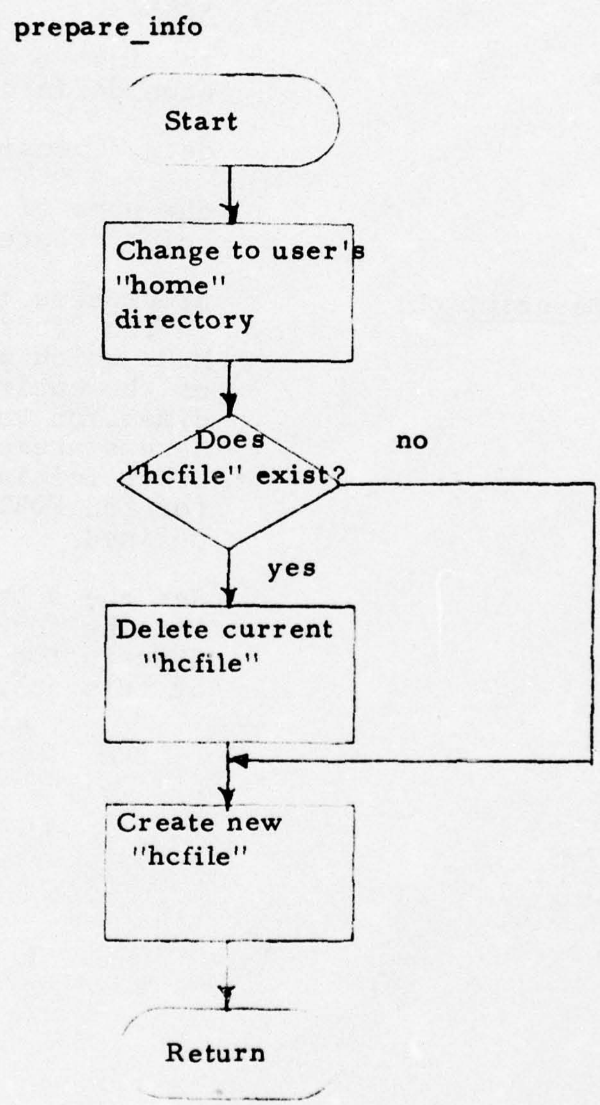
See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: prepare_info

Calling Sequence: call "prepare_info"

Program Description: This program is used by "clprint" to check for the existence of the "hcfile" and delete it if it exists. Then it creates a new "hcfile", in effect, initiating "hcfile" for each call of "clprint".

Flow Chart:



Internal Subroutine Name: preprocess

Calling Sequence: call preprocess (logicptr, lastlog, ndim, file)

Input Parameters:

logicptr — pointer to the MOOS logic file (ptr)

lastlog — the last logic node to have ever been defined (fixed (35))

ndim — data dimensionality (fixed (35))

file — the name of the FORTRAN program being created (char (168))

Program Description: preprocess is the subroutine in the "fortlogc" option of MOOS which performs a preprocess of the entire logic tree to dimension variables, create data statements, and perform other initialization procedures for the FORTRAN program to be defined

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

MOOS Function Name: probconf

MOOS Function Number: 28

Calling Sequence: Type in "probconf [(treename)] [(nodename)]"

Input Parameters: Standard optional data set selection parameters

Output File Settings: The display file is set up in the rank-order display format. In addition, a histogram file is produced for each data class in the selected data set. These files are named as follows: pc || tree character || data class file. (see below for details) The histogram files are described in more detail in section 3.1.

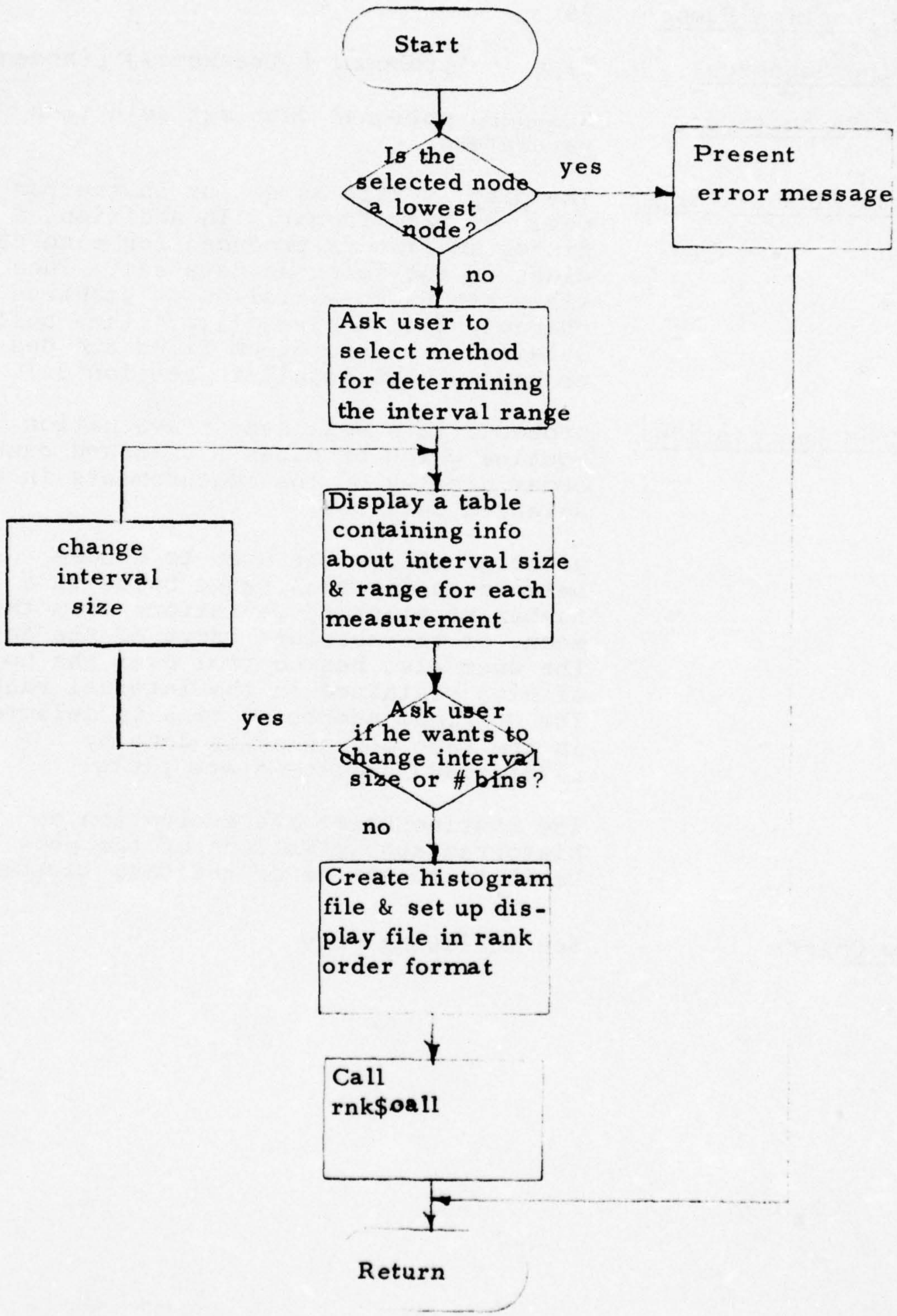
Program Description: probconf is a measurement evaluation routine which produces a standard rank order display of the measurements in the selected data set.

probconf allows the user to choose between an interval range based on a *number of standard deviations from the* mean, or the absolute range of the data. The user also has control over the no. of bins contained in the interval range. The original number of bins is selected in the same manner as is done by ss$display1 for one-space plots.

The routine bases the evaluation on histogram approximations of the probability functions of the data classes.

Flow Chart: **See following page**

probconf



Start
Is the selected node a lowest node?
yes
Present error message
no
Ask user to select method for determining the interval range
Display a table containing info about interval size & range for each measurement
change interval size
Ask user if he wants to change interval size or # bins?
yes
no
Create histogram file & set up display file in rank order format
Call rnk$call
Return

pc tree character data class File Format

1	H1(1)
2	H2(1)
3	H3(1)
	⋮
	H1(ndim)
	H2(ndim)
3*ndim	H3(ndim)
3*ndim+1	H4(1, 1)
	⋮
	H4(nbin, 1)
	H4(1, 2)
	⋮
3*ndim+(nbin*ndim)	H4(nbin, ndim)

where,

nbin = number of bins
ndim = number of dimensions
H1 = number of bins for each measurement
H2 = lower bound of the interval range for each measurement
H3 = upper bound of the interval range for each measurement
H4 = histogram data(all counts are divided by the total number of vectors, therefore can be considered probabilities.

Internal Subroutine Name: ptwospace

Calling Sequence: call ptwospace (logicptr, pairptr, reject)

Input Parameters:

logicptr — pointer to MOOS logic file (ptr)

pairptr — pointer to logic for this pair (ptr)

reject — reject logic node number (fixed (35))

Program Description: ptwospace is the subroutine in the "fortlogc" option of MOOS which is the executive for creating FORTRAN code for a two-space projection in a pairwise node.

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: ptwospace$two

Calling Sequence: call ptwospace$two (logicptr, pairptr, reject)

Input Parameters:

logicptr — pointer to MOOS logic file (ptr)

pairptr — pointer to logic for this class pair (ptr)

reject — reject logic node number

Program Description: ptwospace$two is the subroutine in the "fortlogc" option of MOOS which creates FORTRAN code for any two-space projection in a pairwise node

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: pwfisher

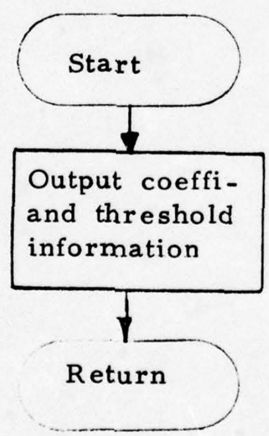
Calling Sequence: call pwfisher (critptr, numdim, numbound, ifile)

Input Parameters:

- critptr - pointer to logic block
- numdim - number of dimensions
- numbound - value contained in third quarter of first word in logic block
- ifile - output file name

Program Description: pwfisher outputs the fisher coefficients, the discriminant coefficients, the five thresholds available, and the (numbound) threshold(s) used for a fisher pair.

Flow Chart:



Internal Subroutine Name: pwlogic

Calling Sequence: call pwlogic (fileptr, optfile, numdim, dex, nodes, lnn, lcc, ifile)

Input Parameters:

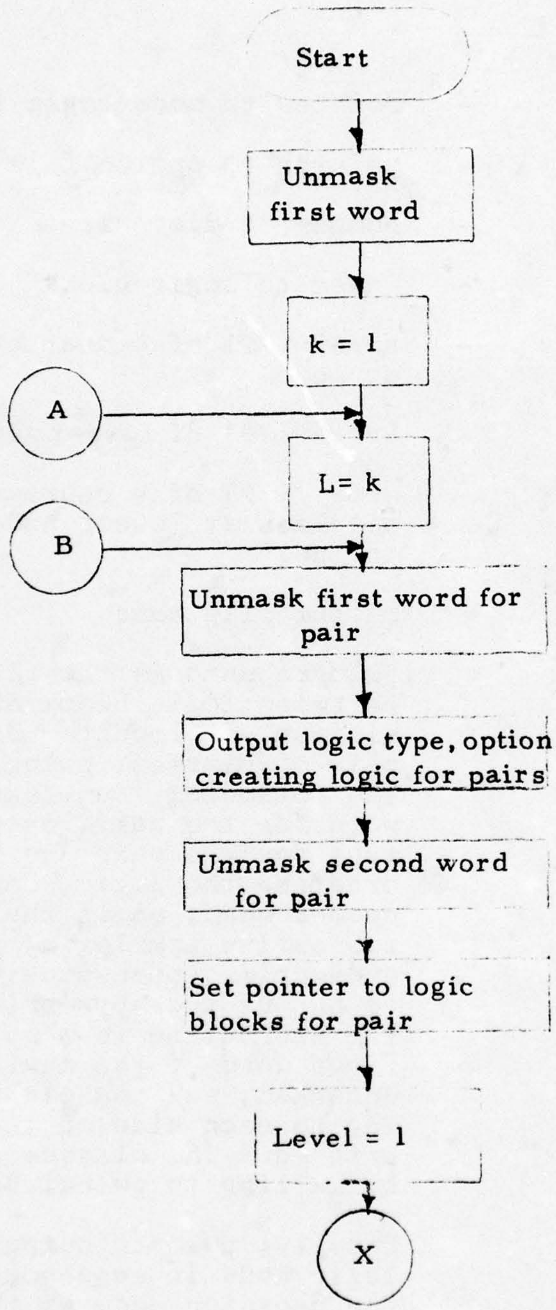
- fileptr - pointer to moos logic file
- optfile - pointer to option_file
- numdim - number of dimensions
- dex - index to logic block
- nodes - array (72) of 4 character node names at node
- lnn - array (72) of lowest logic nodes
- lcc - array (72) of 4 character node names of class at lowest node (parallel to lnn)
- ifile - output file name

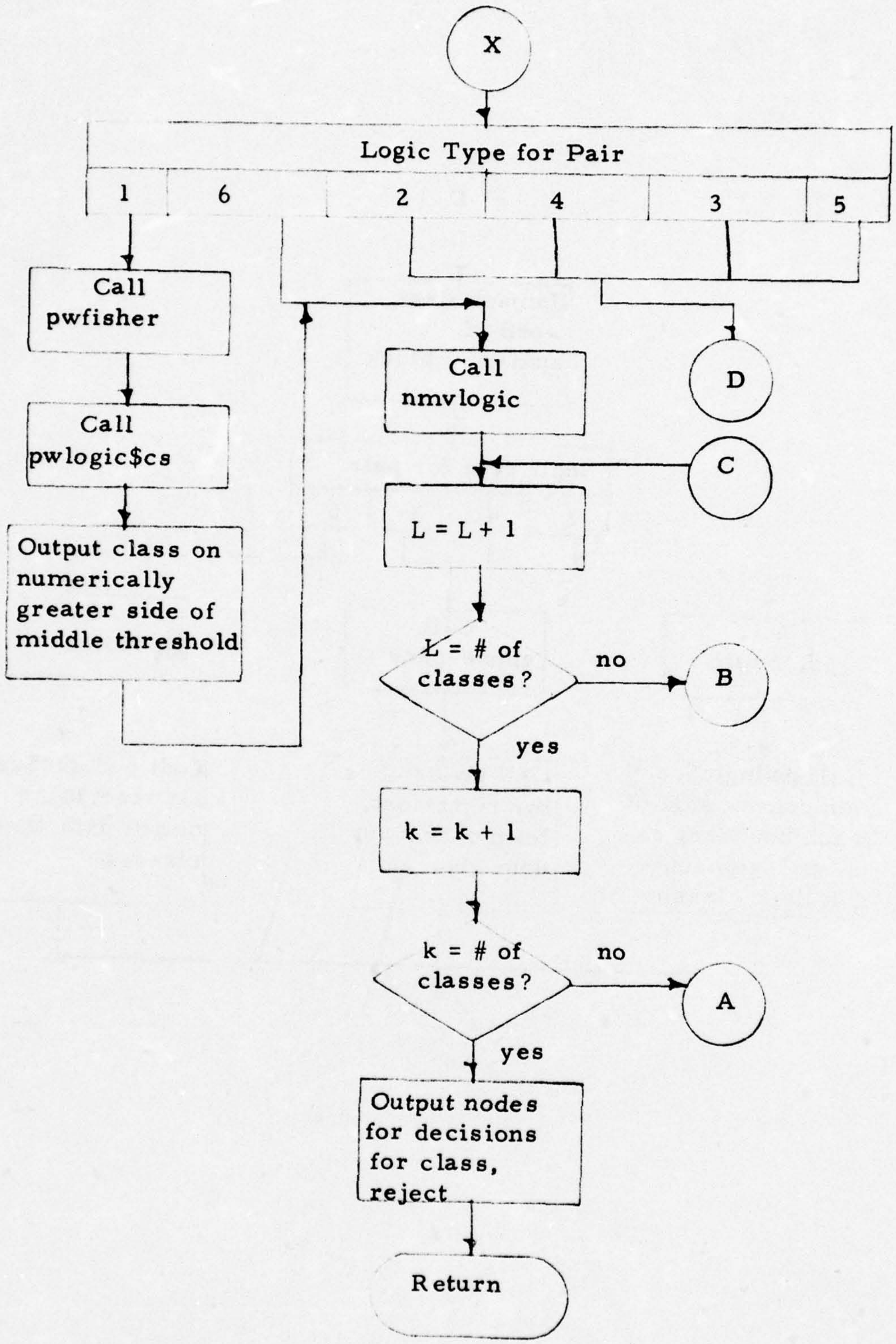
Program Description: pwlogic unmasks the first word of the pairwise logic block and outputs the minimum vote count. Then, for each pair of classes, pwlogic performs the following: unmasks the first word for the pair, outputs the logic type for the pair and the option creating the logic, unmasks the second word, calls the apppropriate subroutine (nmvlogic, pwfisher, gpdiscrim, gponespace, or gpboolean) to output the appropriate values. If the subroutine is a gp routine, the first word of the auxillary block is unmasked, and the classes corresponding to each side of the boundary are printed. The classes are obtained by calling to pwlogic$cs.

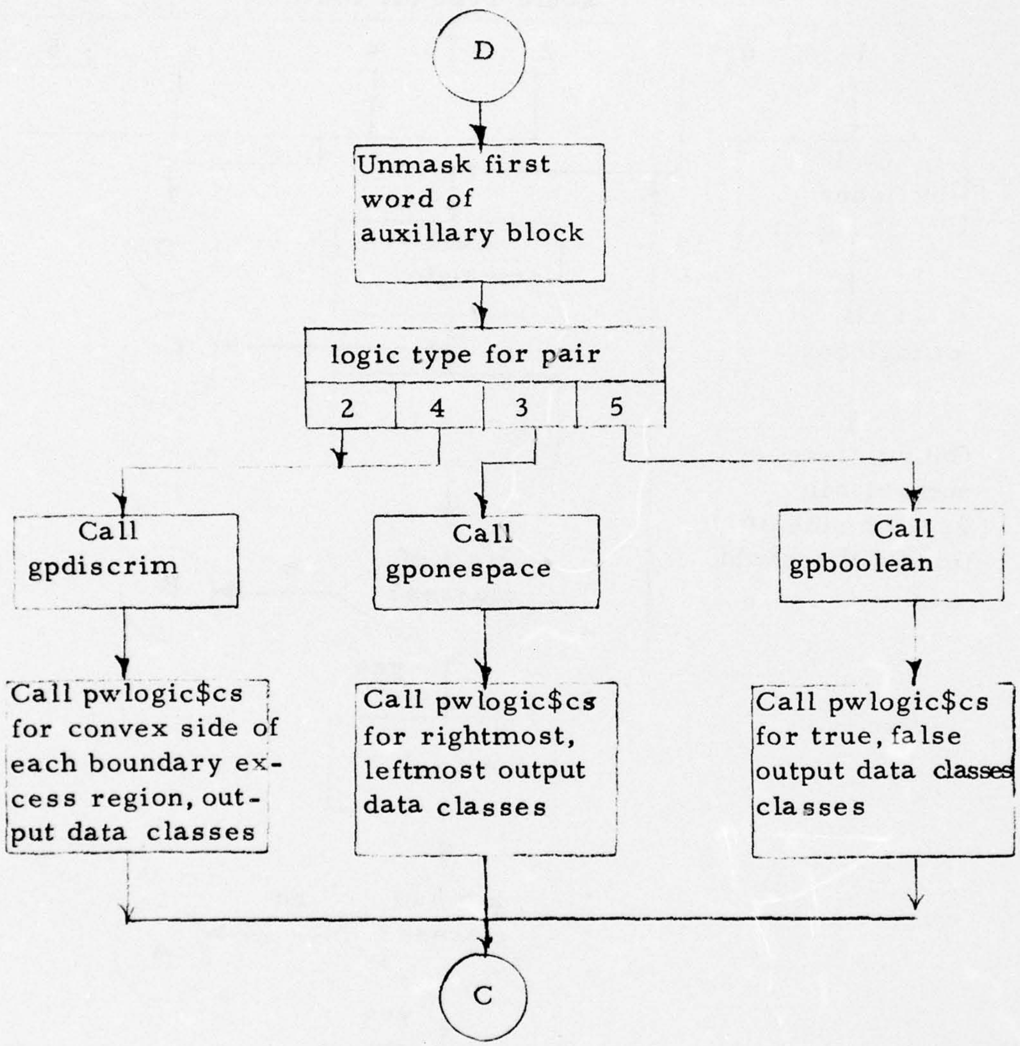
Finally, pwlogic outputs to the next logic node in sequence depending on the decision made at the node.

Flow Chart: See following page.

pwlogic


Start
Unmask first word
k = 1
A
L= k
B
Unmask first word for pair
Output logic type, option creating logic for pairs
Unmask second word for pair
Set pointer to logic blocks for pair
Level = 1
X


X
Logic Type for Pair
1
6
2
4
3
5
Call pwfisher
Call pwlogic$cs
Output class on numerically greater side of middle threshold
Call nmvlogic
D
C
L = L + 1
L = # of classes?
no
B
yes
k = k + 1
k = # of classes?
no
A
yes
Output nodes for decisions for class, reject
Return


D
Unmask first word of auxillary block
logic type for pair
2
4
3
5
Call gpdiscrim
Call gponespace
Call gpboolean
Call pwlogic$cs for convex side of each boundary excess region, output data classes
Call pwlogic$cs for rightmost, leftmost output data classes
Call pwlogic$cs for true, false output data classes classes
C

Internal Subroutine Name: pwlogic$cs

Calling Sequence: call pwlogic$cs (m, L1, L2, L3, nl)

Input Parameters:

m - index of L3 to be matched

L1 - array (72) of lowest logic nodes

L2 - array (72) of 4 character node names of class at lowest node (parallel to L1)

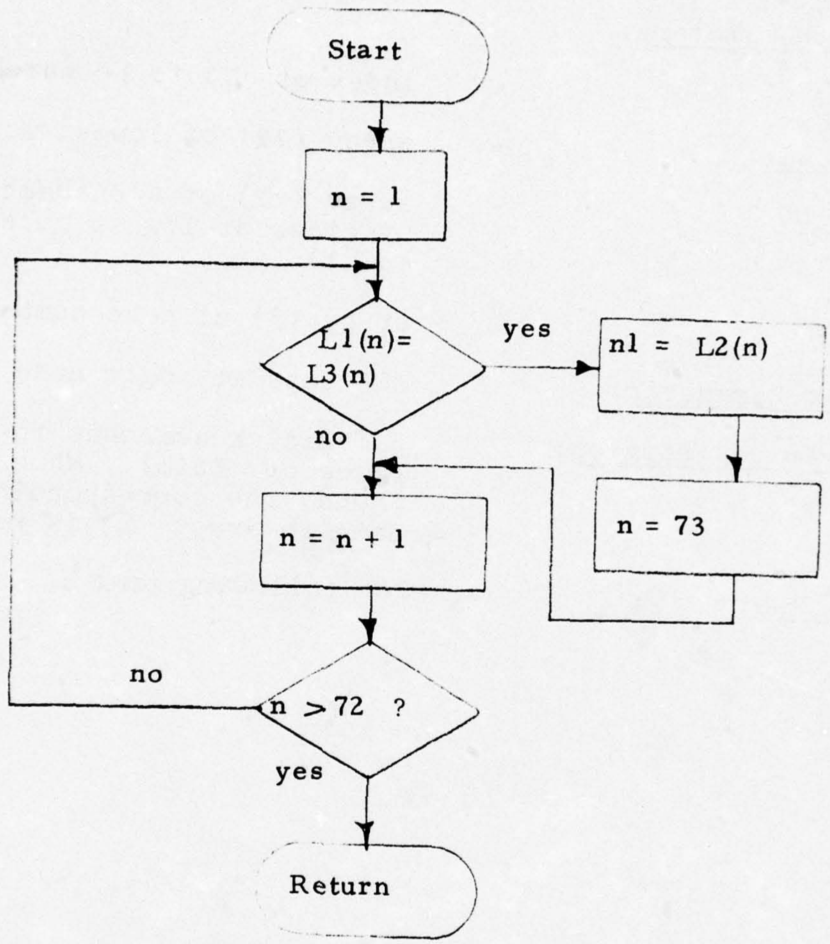
L3 - array (3) of node numbers

Output Parameter: nl-class at logic node L3(m)

Program Description: pwlogic$cs searches the 1 array value of L3(m). When this value is found, the corresponding data class name in array L2 is returned in nl.

Flow Chart: See following page

pwlogic$cs

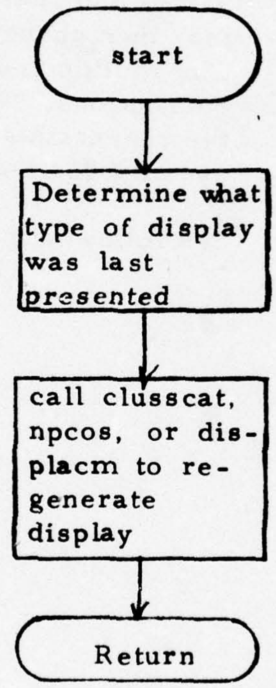


Utility Function Name: rdisplay

Calling Sequence: type in "rdisplay"

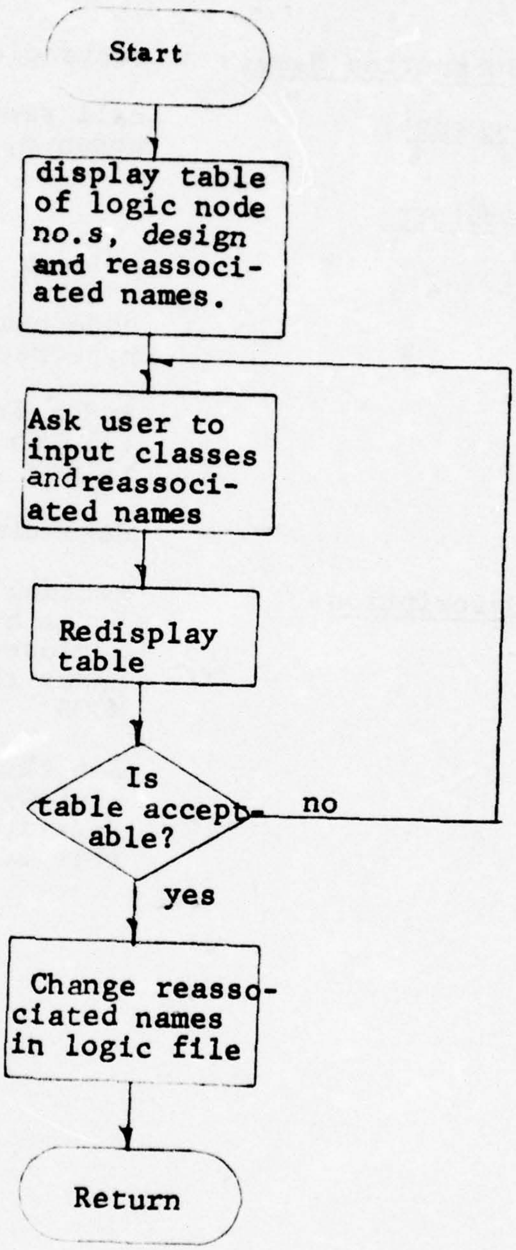
Program Description: rdisplay regenerates the most recent two-space, one-space, or confusion matrix display through calls to clusscat, npcos, or displacm.

Flow Chart:



Utility Function Name:	reasname
Calling Sequence:	Type in "reasname |(treename) (nodename) "
Input Parameters:	Standard optional data set selection parameters
Output File Settings:	reasname changes "reassociated" names in the mooslogic file.
Program Description:	reasname displays a table containing lowest logic node numbers, original logic design names and reassociated names. The user may then change any reassociated names. The routine redisplays the table and allows corrections. When the user is satisfied, the reassociated names in the mooslogic file are changed and the routine exits.
Flow Chart:	See following page.

reasname


Start
display table of logic node no.s, design and reassociated names.
Ask user to input classes and reassociated names
Redisplay table
Is table acceptable?
no
yes
Change reassociated names in logic file
Return

Internal Subroutine Name: rectangle

Calling Sequence: call rectangle (logicptr, nodenum, index, ndim)

Input Parameters:

logicptr — pointer to MOOS logic file (ptr)

nodenum — node number of logic having the hyperrectangle (fixed (35))

index — index from beginning of logic file to logic for this region (fixed (35))

ndim — data dimensionality

Program Description: rectangle generates FORTRAN code for a hyperrectangular region of a closed decision logic node under the "fortlogc" option of MOOS

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Utility Function Name: redraw

Calling Sequence: type in "redraw"

Input File Settings: Either two-space or one-space display files must be set up and boundaries must exist.

Program Description: redraw reconstructs a boundary(or boundaries) which the user has drawn after a new display has been put up.

redraw contains five short subroutines in the form of entry points to perform certain tasks:

redraw$xint returns the intersection point of any line with a given horizontal line.

redraw$yint returns the intersection point of any line with a given vertical line.

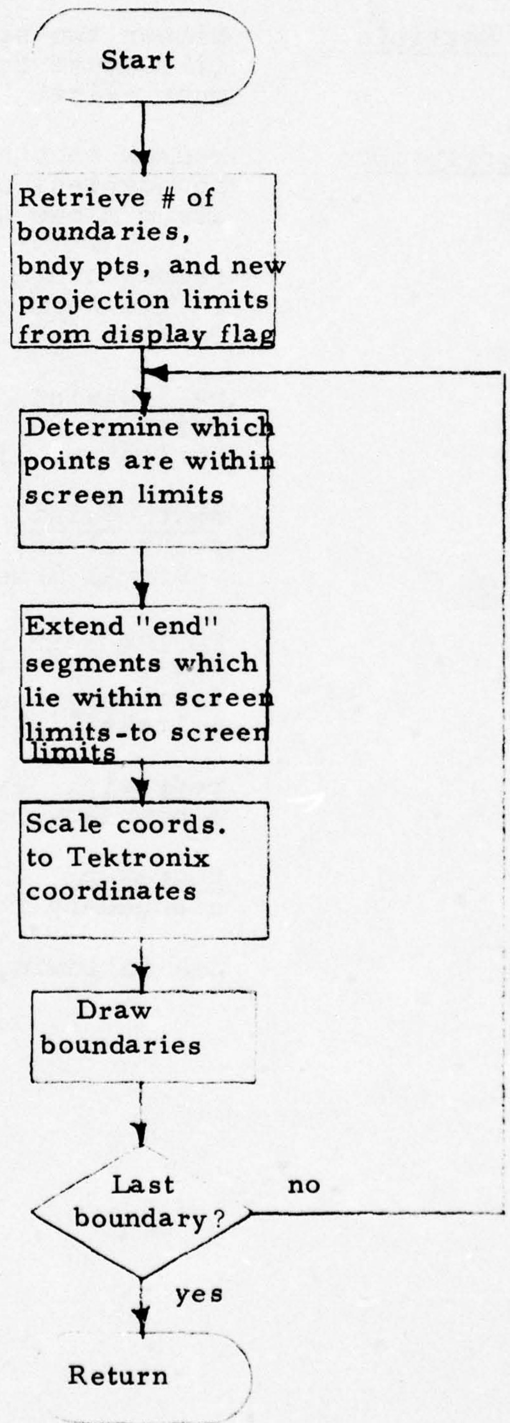
redraw$line draw scales the calculated projection plane coordinates to tektronix coordinates and calls multeks$line to draw lines.

redraw$p1 extends the first segment in a boundary to the edge of the screen.

redraw$p2 extends the last segment in a boundary to the edge of the screen.

Flow Chart: See following page

redraw


Start
Retrieve # of boundaries, bndy pts, and new projection limits from display flag
Determine which points are within screen limits
Extend "end" segments which lie within screen limits-to screen limits
Scale coords. to Tektronix coordinates
Draw boundaries
Last boundary?
no
yes
Return

Utility Function Name: remtree

Calling Sequence: type in "remtree(treename/"all")"

Input Parameters:

treename	specify a particular data set
"all"	perform operation for all data sets in "trandata".

Output File Settings:

tables	trandata's "seg_o_trees" will be reduced
segments	trandata's "structure" will be reduced

Program Description: This routine calls s_p$trem and returns control to the user. The files "seg_o_trees" and "structure" will reflect the deletions of data sets.

Utility Function Name: restore

Calling Sequence: type in "restore(treename/"all")"

Input Paramenters:

treename — specify a particular data set

"all" — perform operation for all data sets in the user's directory, "saved_trees"

Output File Settings: The "sysdata" file is updated to reflect the new additions.

Program Description: The program calls s_p$trst and returns control to the user.

Utility Function Name: restorec

Calling Sequence: type in "restorec(treename/"all")"

Input Parameters:

treename — specify a particular data set

"all" — perform operation for all data sets in "trandata".

Output File Settings: The "sysdata" file is updated to reflect the new additions.

Program Description: This routine calls s_p$tout and returns control to the user.

Utility Function Name: restruct

Calling Sequence: type in "restruct [(treename)]"

Input Parameters:

treename — char(8) treename of displayed data set

Input File Settings: display and csdata must be set up for a one or two space plot and at least one boundary must have been placed in the data file

Output File Settings:

dataclass files: Two or three new data class files are created (depending on the no. of boundaries). If there are no errors, the old data class file is deleted.

treename — Mean and covariance entries are added for each new data class.

sysdata — sysdata is adjusted to reflect the new tree structure.

display — The display file is modified for reprojecting the split classes.

rest file — A temporary file named "rest_file" is created in the user's process directory as a buffer area.

csdata — projection coordinates are moved to match the new data class structure.

Program Description: If the display is one-space, restruct calls internal subroutine ros to restructure the data class.

If the display is two-space, restruct presents a list of data classes displayed and asks the user to choose one. restruct then asks the user to input 2 or 3 unique four character node names. New data class files are then created and the vectors of the data set being divided are sorted according to the drawn boundaries.

If any new data class contains no vectors, restruct will present an error message and exit.

restruct will not divide a class correctly if:

(1) any boundary is not convex
(2) the so-called "convex point" is not on the convex side of a boundary
(3) two boundaries intersect within the range of the vectors being separated.

restruct utilizes two internal entry points:

restruct$side - evaluates the equation of a line for any point and returns the sign of the result

restruct$mncv - computes mean and covariance values for the new data class files

If restruct is called from a nlm display, and if this display was the result of a data reduction or clustering algorithm, cl_restruct is called. cl_restruct moves the restructuring of the displayed data set to the original data set, i.e., both trees are structured based on the boundary(s) drawn on the projection of the clustered tree.

Naming Convention:

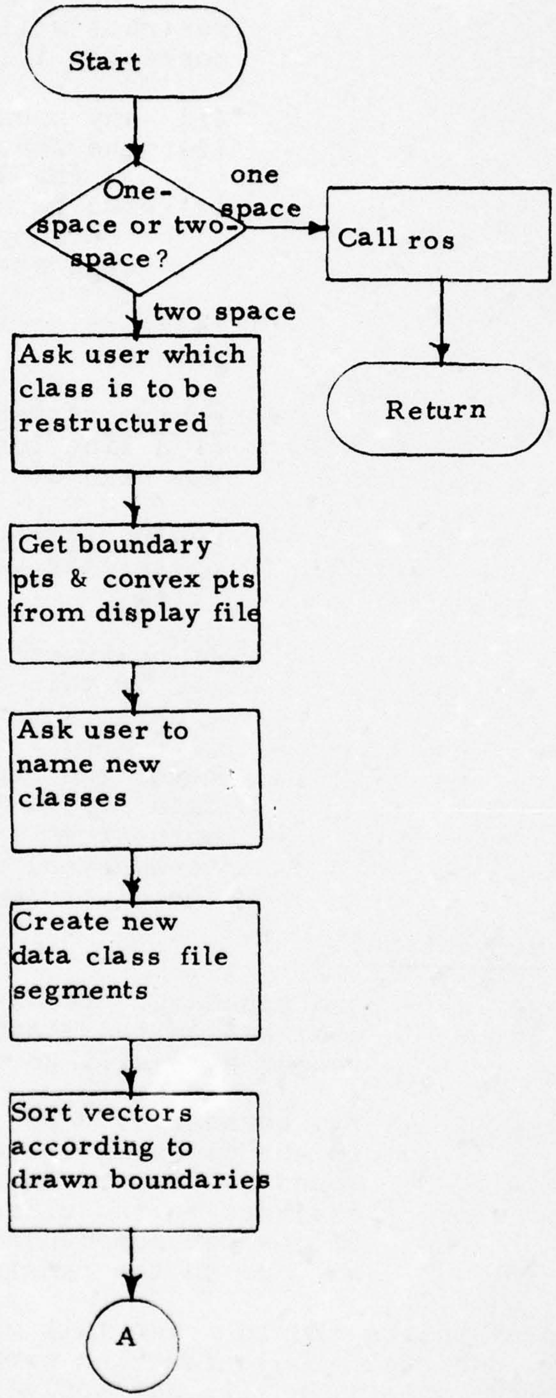
Case I - one boundary. The first name given will be assigned to the class on the convex side. The second name will go to the concave side.

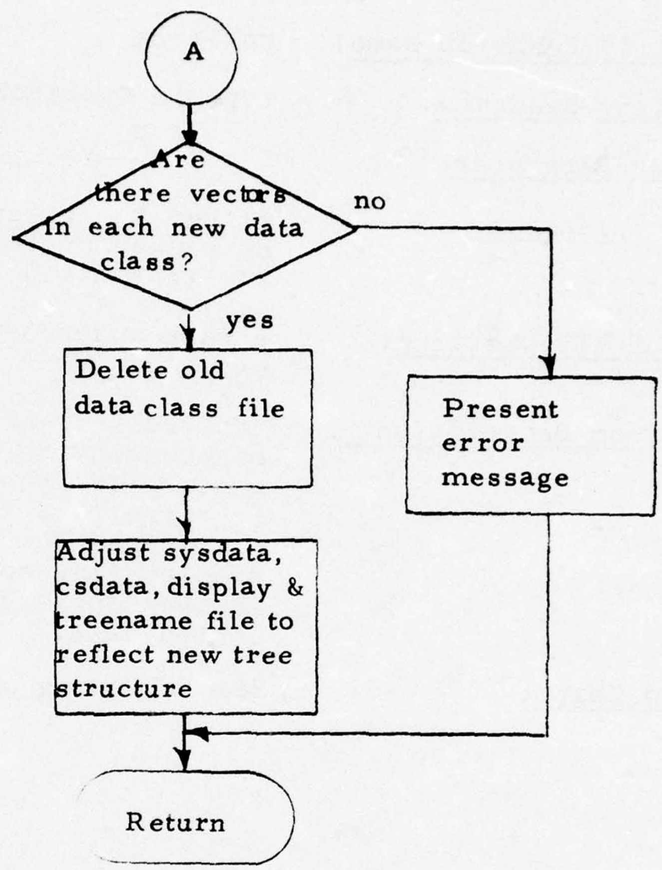
Case II - two boundaries. The first name will be assigned to the class on the convex side of the first boundary drawn. The second name will be assigned to the class on the convex side of the second boundary drawn. The third name will go to the remaining region.

Two convex points should never fall within the same region; however, restruct should function correctly if this case occurs. The naming convention will not apply in this event.

Flow Chart: See following page.

restruct


Start
One-space or two-space?
one space
Call ros
Return
two space
Ask user which class is to be restructured
Get boundary pts & convex pts from display file
Ask user to name new classes
Create new data class file segments
Sort vectors according to drawn boundaries
A


A
Are there vectors in each new data class?
no
yes
Delete old data class file
Present error message
Adjust sysdata, csdata, display & treename file to reflect new tree structure
Return

Utility Function Name: rnk$bcls

Calling Sequence: type in "rnk$bcls (classname)"

Input Parameter:

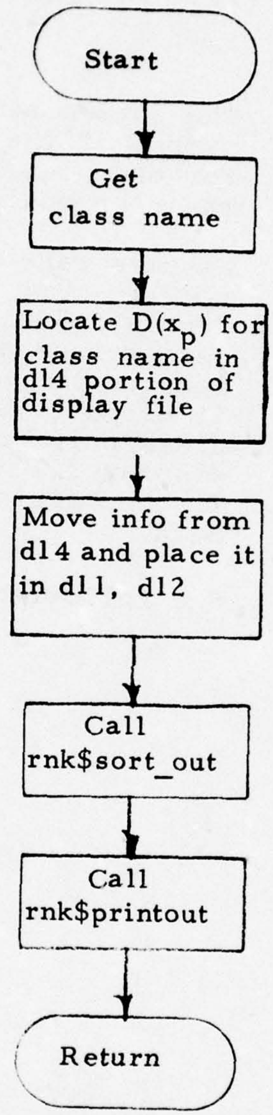
classname — either the entire name of the class to be ranked or the unique class symbol for that class to be ranked

Input File Setting: a rank order display file format is expected

Program Description: "rnk$bcls" ranks an entire class by measurement as a result of discrim measure or probability of confusion. It extracts $D(x_p)$ for the selected class from the dl4 portion of the display file and places it in dl2. It then sorts and prints out the sorted data.

Flow Chart: See following page

rnk$bcls


Start
Get class name
Locate $D(x_p)$ for class name in dl4 portion of display file
Move info from dl4 and place it in dl1, dl2
Call rnk$sort_out
Call rnk$printout
Return

<u>Utility Function Name:</u> rnk$bycp

<u>Calling Sequence:</u> Type in "rnk$bycp(class1, class2)"

<u>Input Parameters:</u>

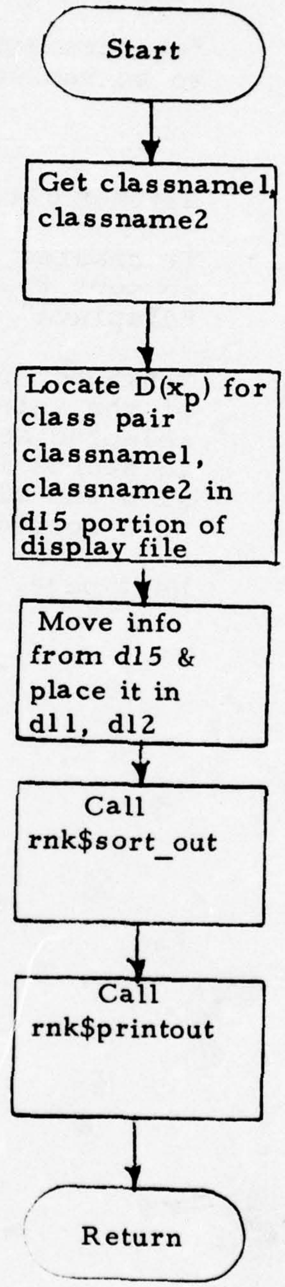
class1, class2 — The entire name or the unique class symbol of the class pair to be ranked. For example, to rank class pair (A,B), the user would enter "rnk$bycp A B".

<u>Input File settings:</u> assumes rank order display file format

<u>Program Description:</u> "rnk$bycp" ranks a class pair by measurement as a result of discrim measure or probability of confusion. It extracts $D(x_p)$ for the selected class pair from the dl5 portion of the display file and places it in dl2. It then sorts and prints out the sorted data.

<u>Flow Chart:</u> See following page

rnk$bycp


Start
Get classname1, classname2
Locate $D(x_p)$ for class pair classname1, classname2 in dl5 portion of display file
Move info from dl5 & place it in dl1, dl2
Call rnk$sort_out
Call rnk$printout
Return

Utility Function Name: rnk$mbc

Calling Sequence: Type in "rnk$mbc(measurement number)"

Input Parameter: "measurement number" is the measurement to be ranked by class

Input File Setting:

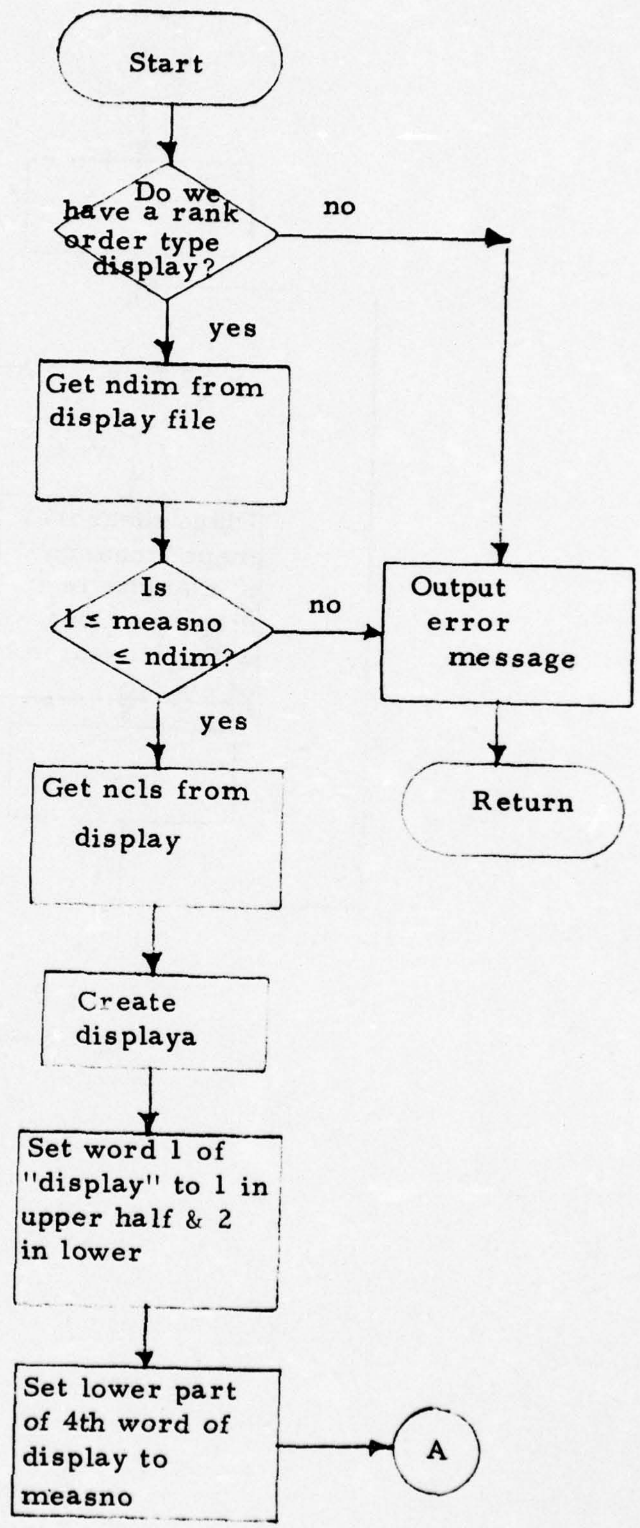
display: assumes rank-order display file format

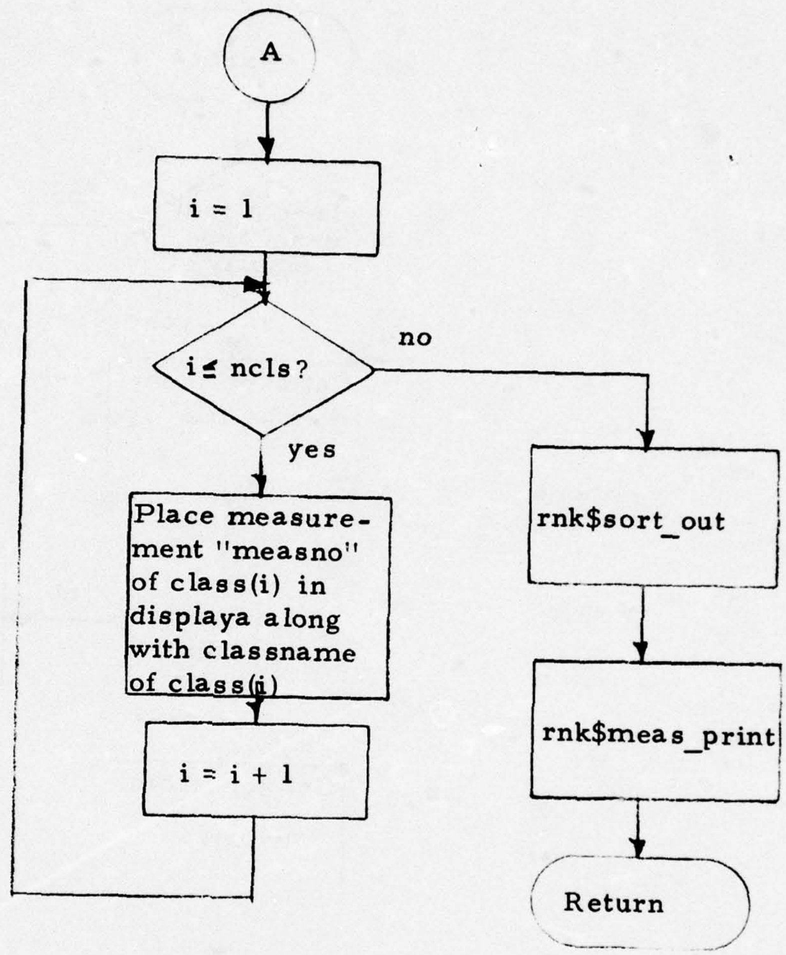
Output File Settings: it creates a file called "displaya" to sort the classes. The format of "displaya" is described elsewhere.

Program Description: "rnk$mbc" prints out a class ranking of the selected measurement. The selected measurement of each class is stored in file "displaya" along with the class name. The file is then sorted and printed out.

Flow Chart: Next page.

rnk$mbc


Start
Do we have a rank order type display?
no
yes
Get ndim from display file
Is 1 ≤ measno ≤ ndim?
no
yes
Output error message
Return
Get ncls from display
Create displaya
Set word 1 of "display" to 1 in upper half & 2 in lower
Set lower part of 4th word of display to measno
A


A
i = 1
i ≤ ncls?
no
yes
Place measurement "measno" of class(i) in displaya along with classname of class(i)
i = i + 1
rnk$sort_out
rnk$meas_print
Return

Utility Function Name: rnk$mbcp

Calling Sequence: Type in "rnk$mbcp(measurement number)"

Input Parameter: "measurement number" is the measurement to be ranked by class pair

Input File Setting:

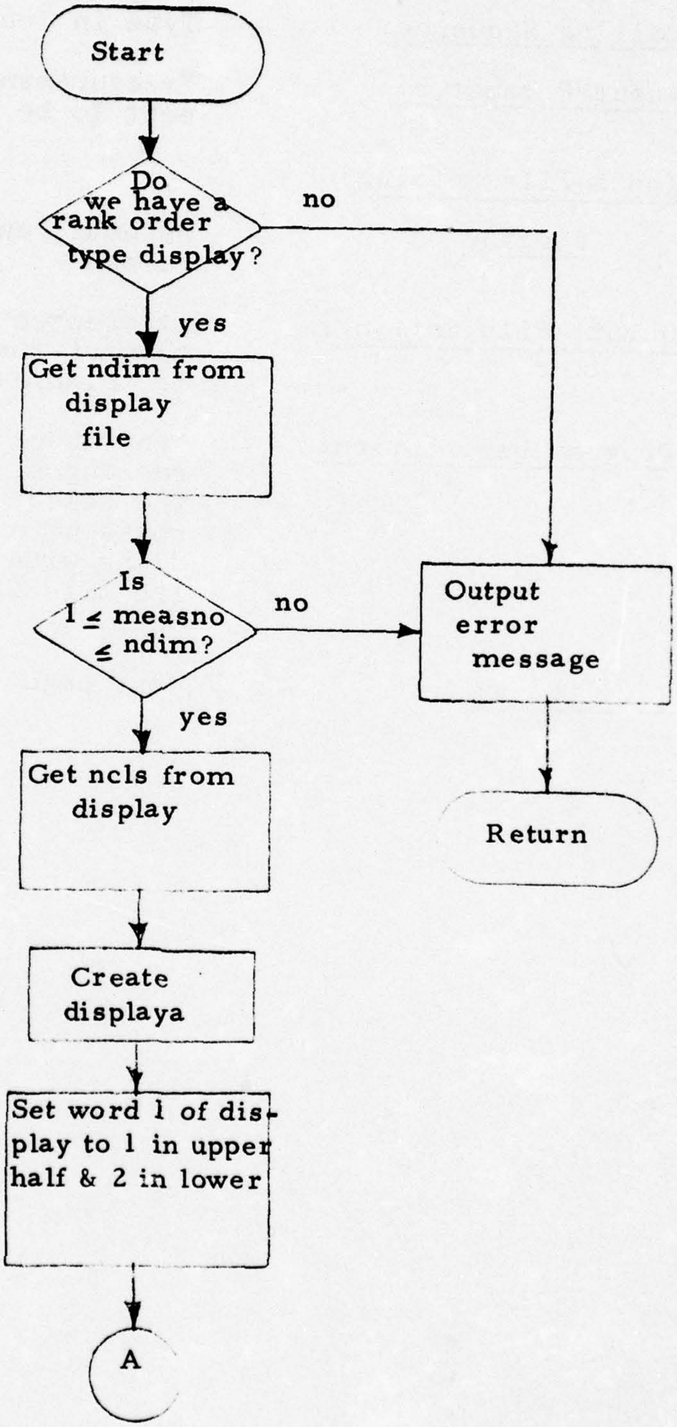
display — assumes rank-order display file format

Output File Setting: it creates a file called "displaya" to sort the class pairs. The format of "displaya" is described elsewhere.

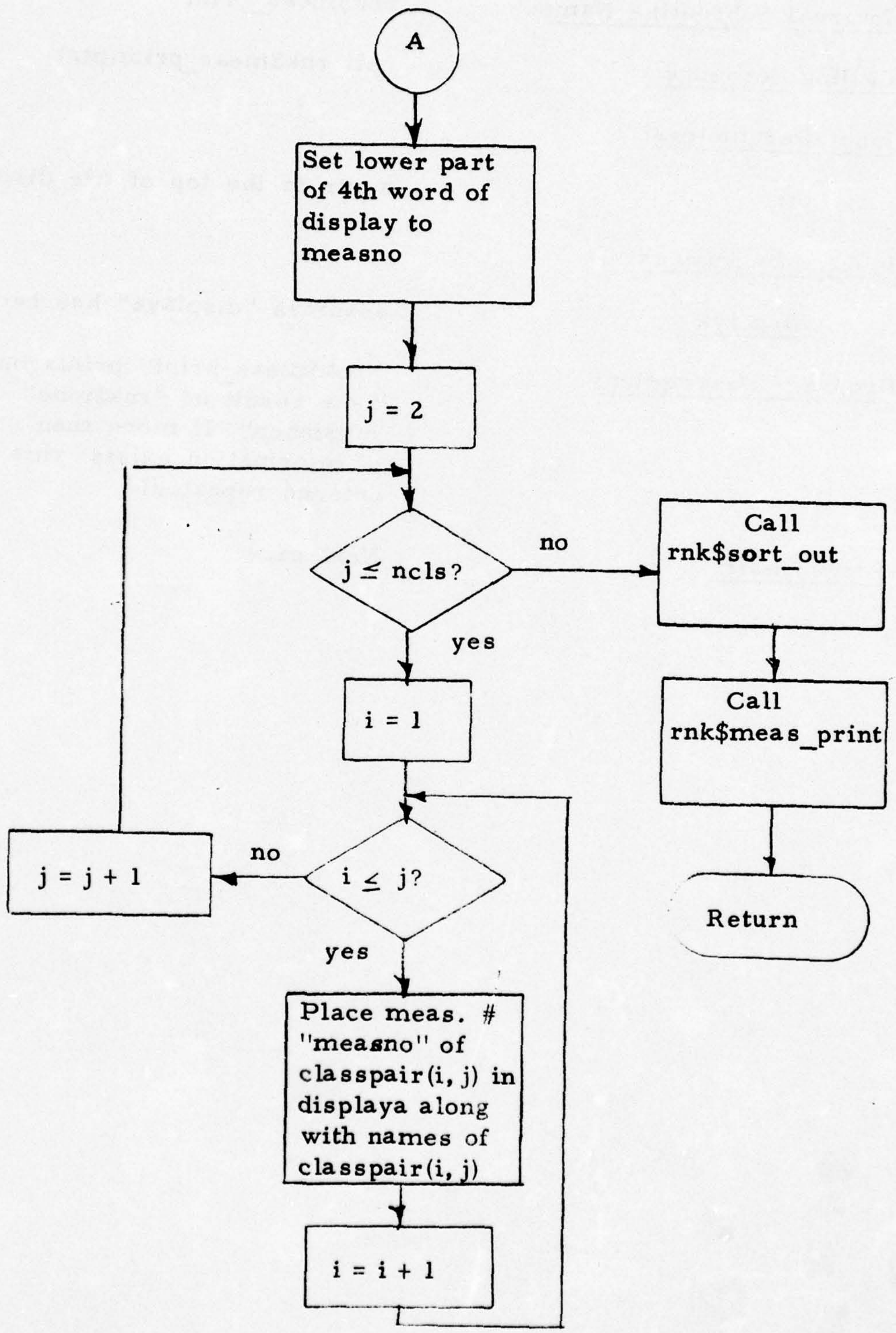
Program Description: "rnk$mbcp" prints out a class pair ranking of the selected measurement. The selected measurement of each class pair is stored in "displaya" along with the class pair name. The file is then sorted and printed out.

Flow Chart: Next page

rnk$mbcp

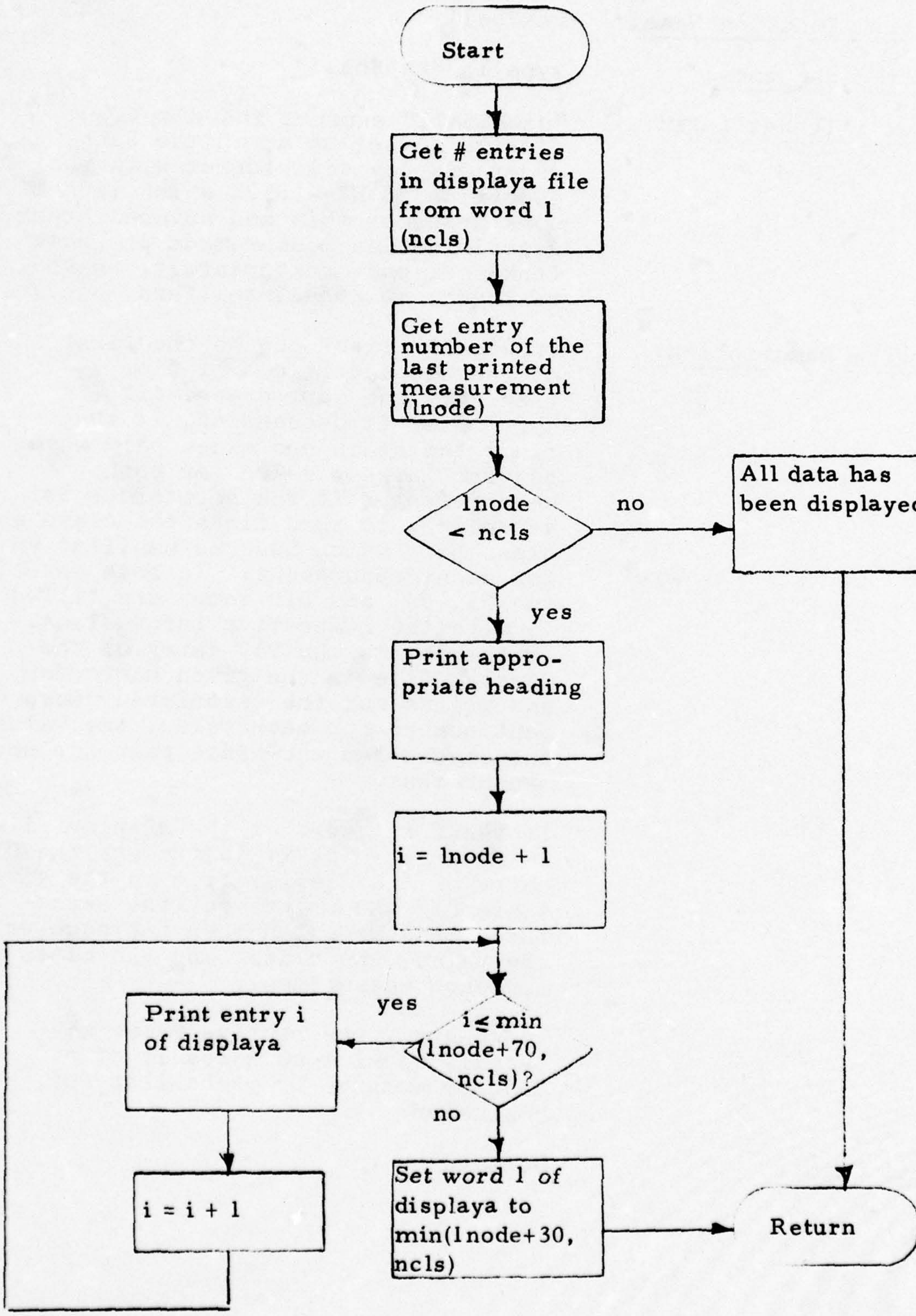


Start
Do we have a rank order type display?
no
yes
Get ndim from display file
Is 1 ≤ measno ≤ ndim?
no
yes
Output error message
Return
Get ncls from display
Create displaya
Set word 1 of display to 1 in upper half & 2 in lower
A


A
Set lower part of 4th word of display to measno
j = 2
j ≤ ncls?
no
yes
Call rnk$sort_out
Call rnk$meas_print
Return
i = 1
i ≤ j?
no
yes
j = j + 1
Place meas. # "measno" of classpair(i, j) in displaya along with names of classpair(i, j)
i = i + 1

Internal Subroutine Name:	rnk$meas_print
Calling Sequence:	call rnk$meas_print(ptr)
Input Parameters:	
ptr	a ptr to the top of file displaya
Input File Settings:	
displaya	assumes "displaya" has been set up
Program Description:	"rnk$meas_print" prints out displaya as a result of "rnk$mbc" or rnk$mbcp." If more than one page of information exists, this routine is entered repeatedly.
Flow Chart:	Next page

rnk$meas_print



Utility Function Name: rnk$oall

Calling Sequence: type in "rnk$oall"

Input File Settings: "rank$oall" expects the display file to be set up as in the Rank Order display file format with the exception of D7 - D12, which is the work area for this and several other discrimination measurement programs (rnk$sort_out, rnk$printout, rnk$bcls, rnk$bycp, sel$meas, sel$thrs, etc.)

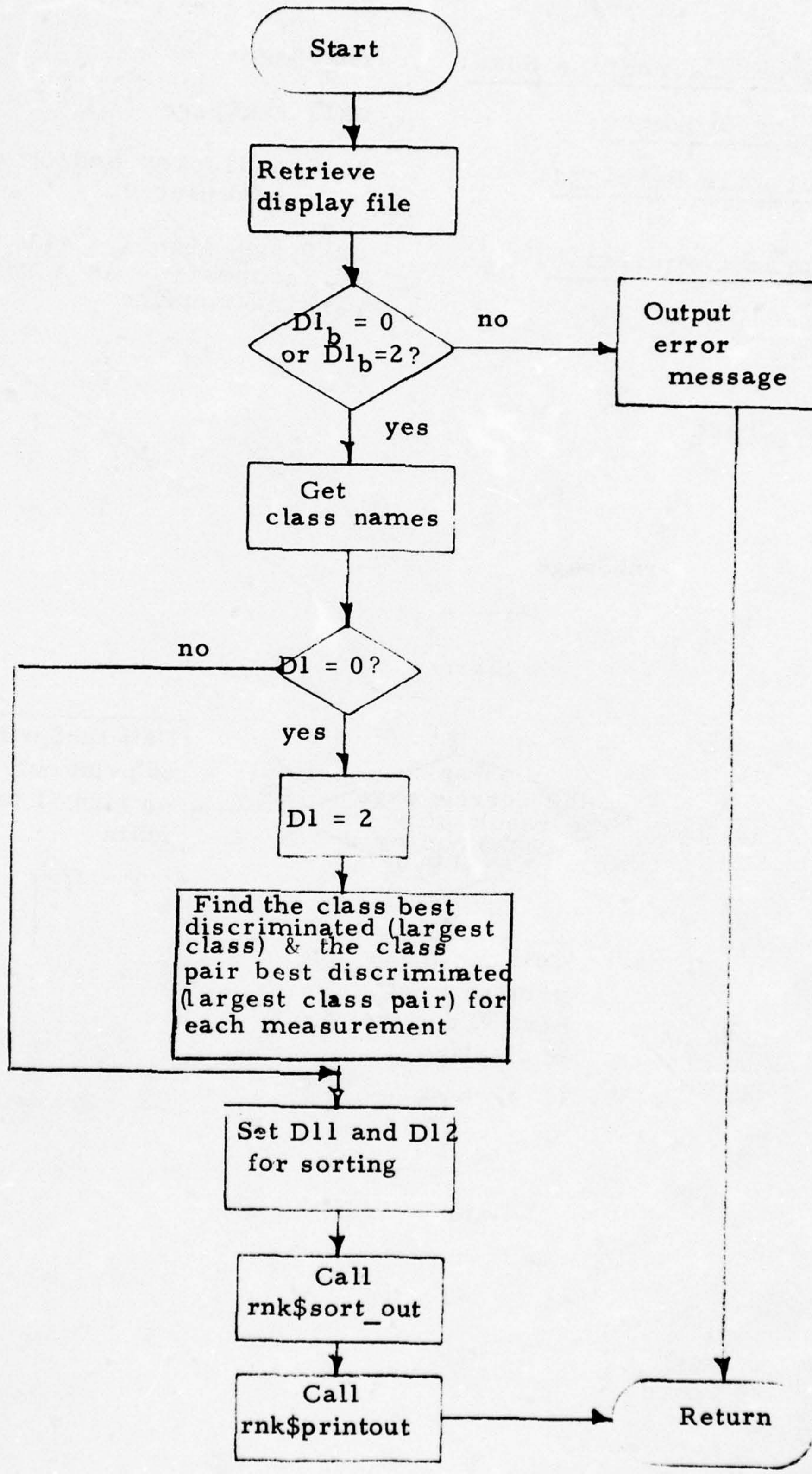
Program Description: "rnk$oall" first checks the first word of the display file. If Ø or 2, it then gets the sort order. If the sort order is descending, it then finds the class and class pair which has the largest value for each measurement. If the sort order is ascending, it then finds the class and class pair which has the smallest value for each measurement. In both cases, the D8, D9, and D10 entry are filled in with the respective information. It then sorts the D13 entry of the display file in the given sortorder and prints out the associated measurement number for each value, the value, the class, and the class pair for each measurement.

If the first word of the display file is other than 0, it just sorts the D13 entry of the display file in the given sortorder and prints out the associated measurement number for each value, the value, the class, and the class pair for each measurement.

In general, the routine gives an overall ranking as a result of a discrim measure or probability of confusion.

Flow Chart: Next page

rnk$oall


Start
Retrieve display file
$Dl_b = 0$ or $Dl_b=2$?
no
Output error message
yes
Get class names
$Dl = 0$?
no
yes
$Dl = 2$
Find the class best discriminated (largest class) & the class pair best discriminated (largest class pair) for each measurement
Set Dl1 and Dl2 for sorting
Call rnk$sort_out
Call rnk$printout
Return

Internal Subroutine Name: rnk$page

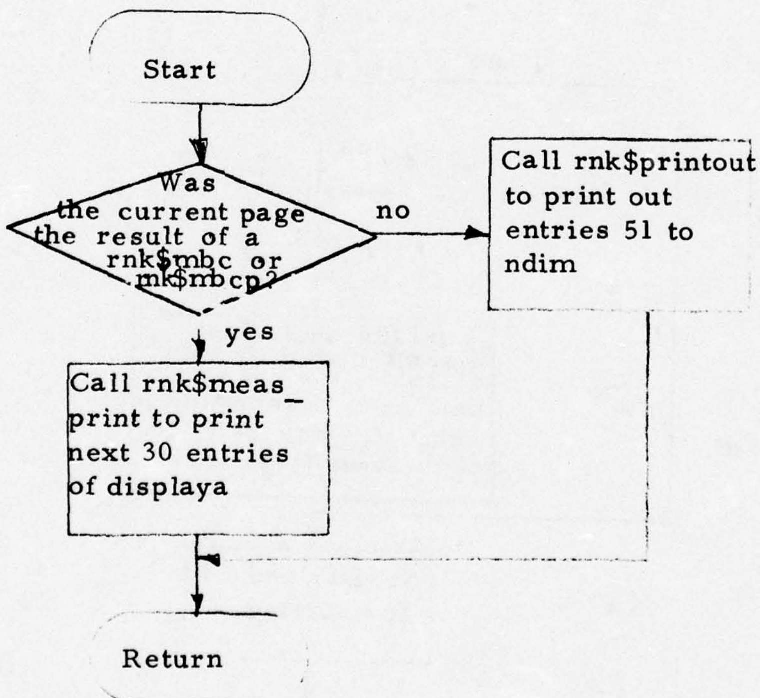
Calling Sequence: call rnk$page

Input File Settings: assume display and/or displaya have been set up

Program Description: rnk$page displays the next page of information as a result of some ranking routine

Flow Chart:

rnk$page



Internal Subroutine Name: rnk$printout

Calling Sequence: call rnk$printout (d77ptr, d111ptr, init, last, type)

Input Parameters:

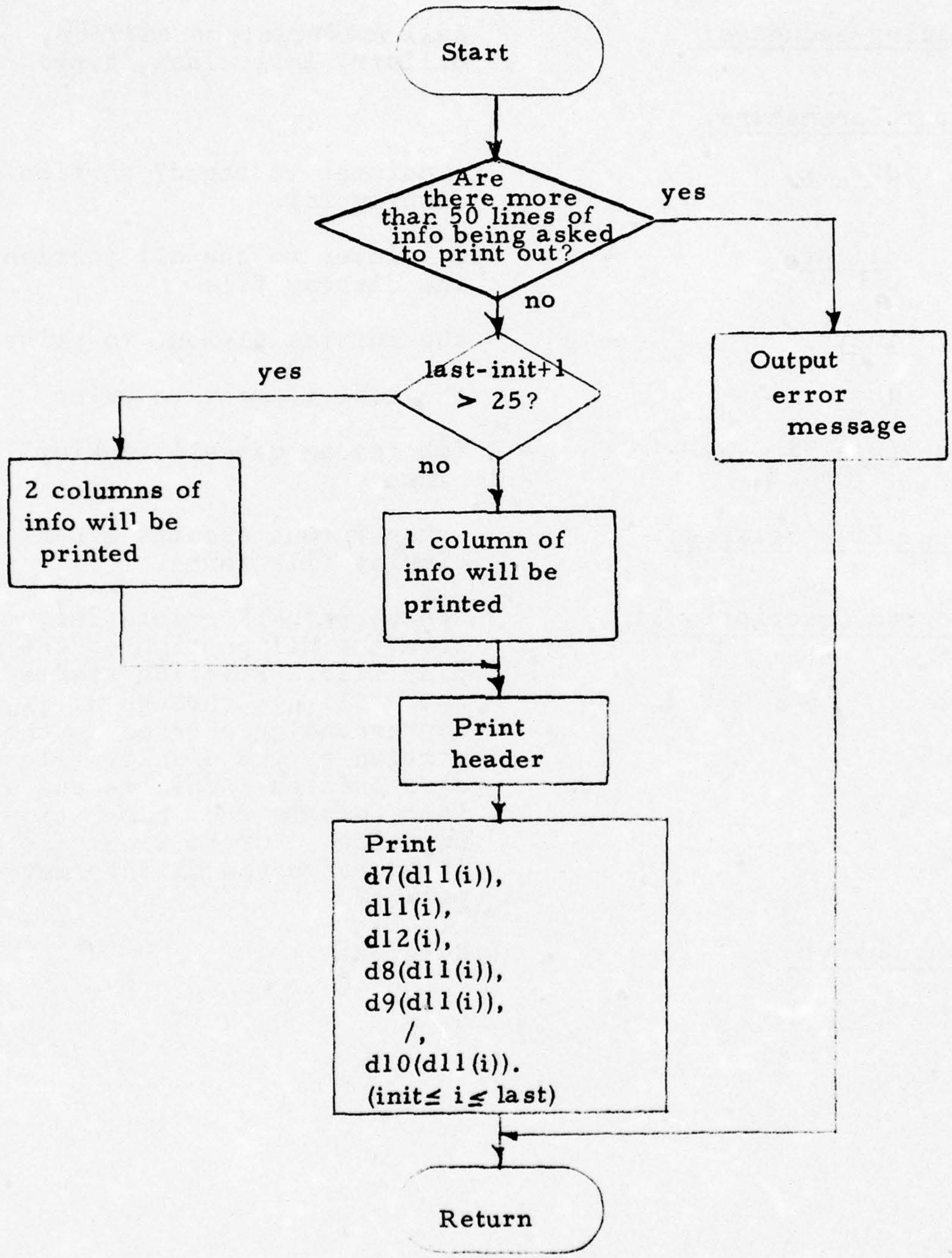
- d77ptr - a pointer to the d7 portion of the display file
- d111ptr - a pointer to the d11 portion of the display file
- init - the initial element to print
- last - the last element to print
- type - = 1 implies an overall ranking; ≠ 1 other

Input File Settings: rnk$printout assumes a rank order display file format

Program Description: "rnk$printout" prints information from the d11 portion of the display file. Printing starts from entry $d11_{init}$ through $d11_{last}$. Corresponding entries in the d7 portion of the display file are also printed. This is the entry that formats the information for printing. Up to 2 columns and 25 lines/column of information is printed.

Flow Chart: Next page.

rnk$printout



Internal Subroutine Name: rnk$sort_out

Calling Sequence: call rnk$sort_out (dlllptr, sort, dim)

Input Parameters:

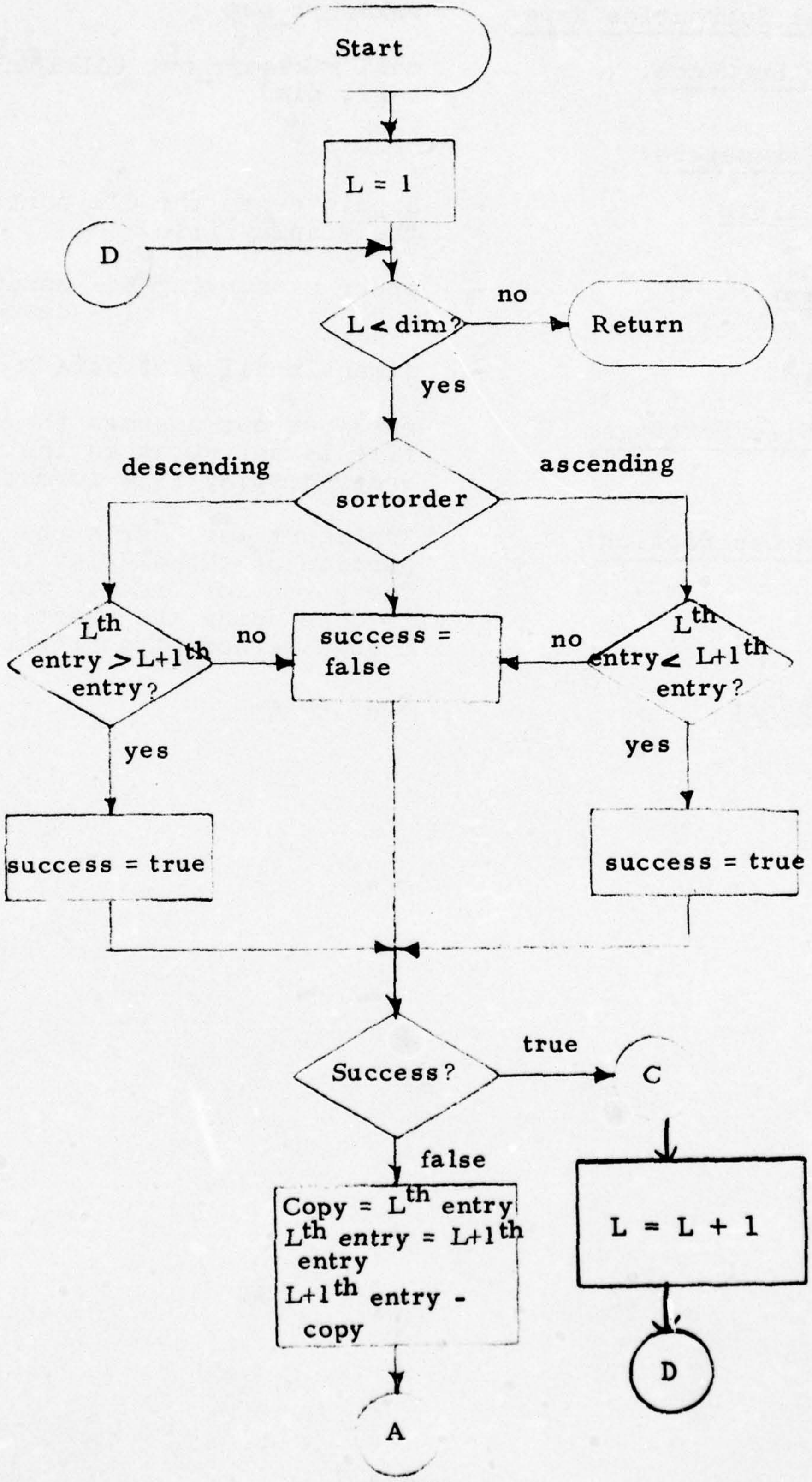
- dlllptr - a pointer to the dll portion of the display file
- sort - order of sorting 0 - ascending
  1 - descending
- dim - dimensionality of data

Input File Settings: rnk$sort_out assumes the display file is set up as in the rank order display file format

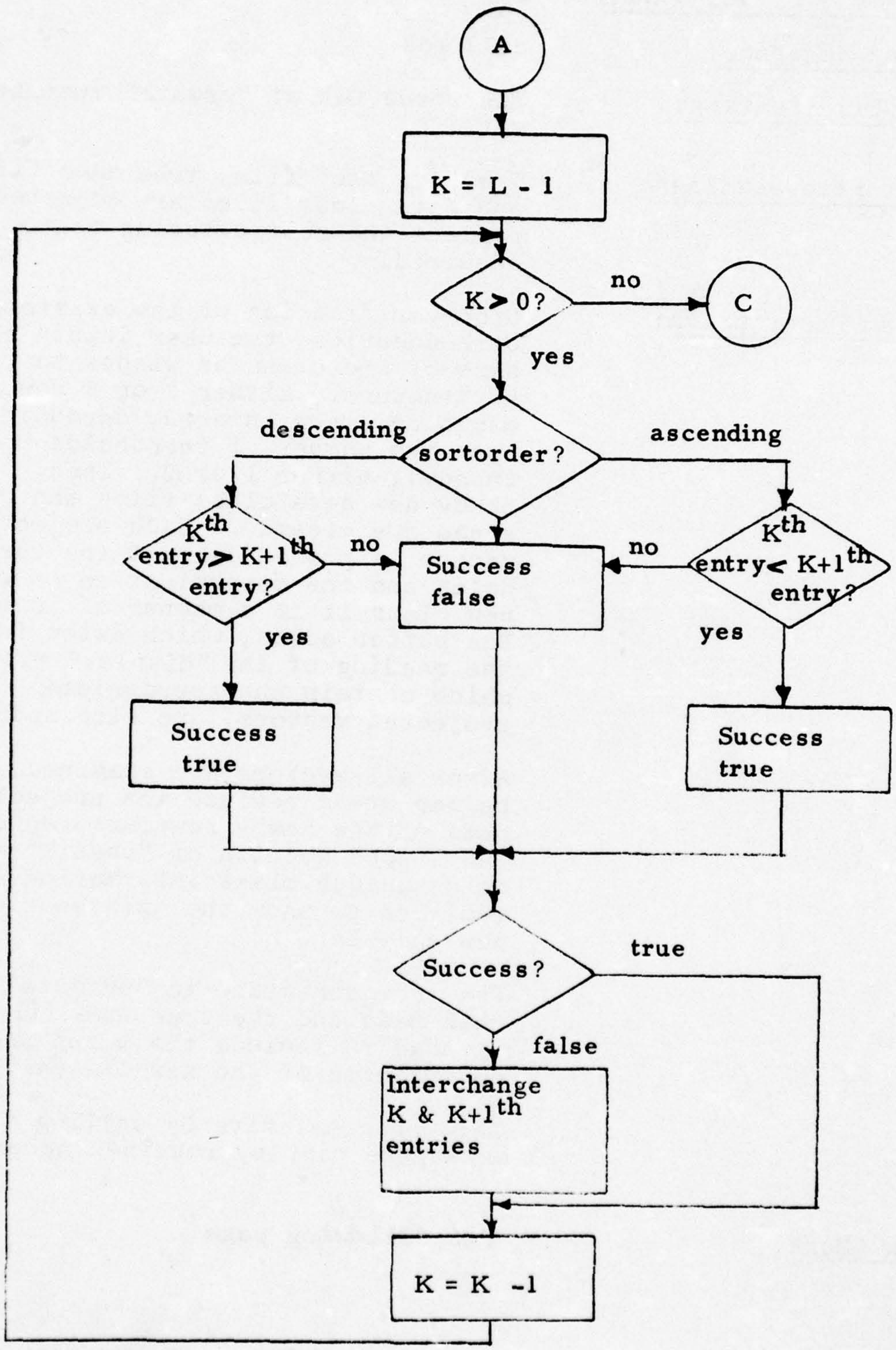
Program Description: "rnk$sort_out" sorts the dll, dl2 portion of the display file in the given sortorder. Sorting is done using the shuttle interchange method of sorting.

Flow Chart: Next page.

rnk$sort_out



Start
L = 1
D
L < dim?
no
Return
yes
descending
sortorder
ascending
Lth entry > L+1th entry?
no
success = false
no
Lth entry < L+1th entry?
yes
success = true
yes
success = true
Success?
true
C
false
Copy = Lth entry
Lth entry = L+1th entry
L+1th entry - copy
L = L + 1
D
A


A
K = L - 1
K > 0?
no
C
yes
descending
sortorder?
ascending
K th entry > K+1 th entry?
no
Success false
no
K th entry < K+1 th entry?
yes
yes
Success true
Success true
Success?
true
false
Interchange K & K+1 th entries
K = K -1

Internal Subroutine Name: ros

Calling Sequence: call ros

Input File Settings: The words D14 of "csdata" must be set.

Output File Settings: The "sysdata" file, tree name file, and data class files are adjusted to reflect the restructuring that occurred.

Program Description: Upon verification of the existence of boundaries, the user inputs the name of the class he wishes to restructure. Either 2 or 3 new class names are then entered, depending upon the number of thresholds (nbndy), either 1 or 2. Then, nbndy new data class files and buffer areas are created. Each projected vector is passed against the thresholds and the determination of which new class it is a member of is made. The buffer areas, which exist for the reading of the "display" file and which contain the appropriate projected vectors, are also updated.

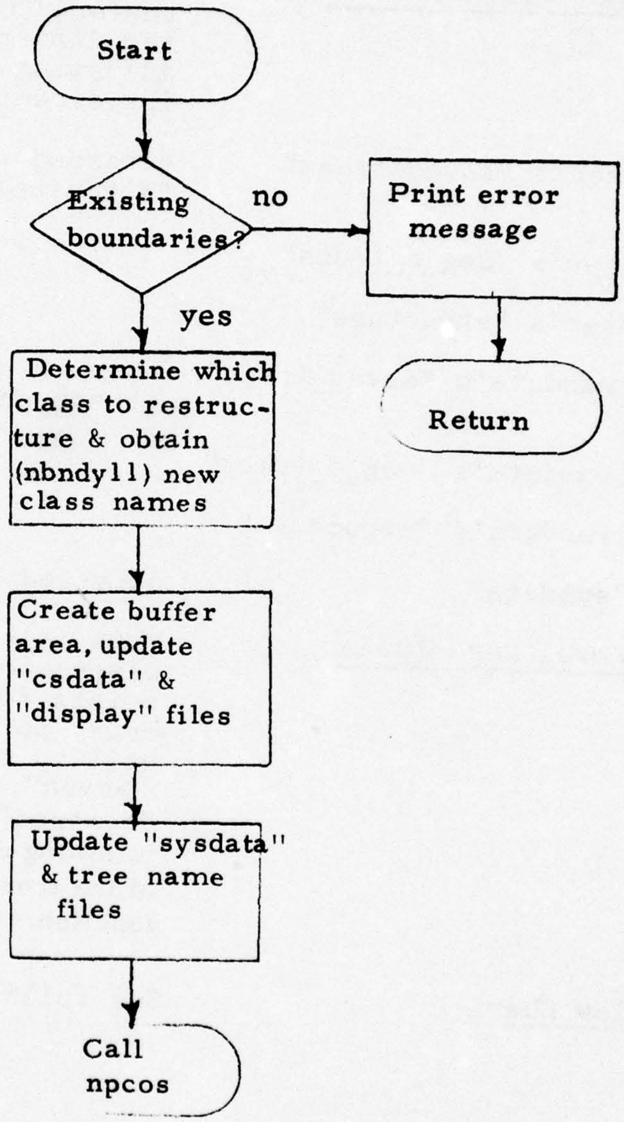
After all vectors are examined, these buffer areas replace the projected data of the newly restructured class. The 4 word section of "csdata" which holds unique class information is modified to show the existence of the new classes.

The tree structure in "sysdata" is then made and the tree name file is altered to include the means and covariances of the new classes.

This program exits by calling the one-space display routine "npcos".

Flow Chart: **See following page**

ros



Start
Existing boundaries?
no
Print error message
Return
yes
Determine which class to restructure & obtain (nbndy11) new class names
Create buffer area, update "csdata" & "display" files
Update "sysdata" & tree name files
Call npcos

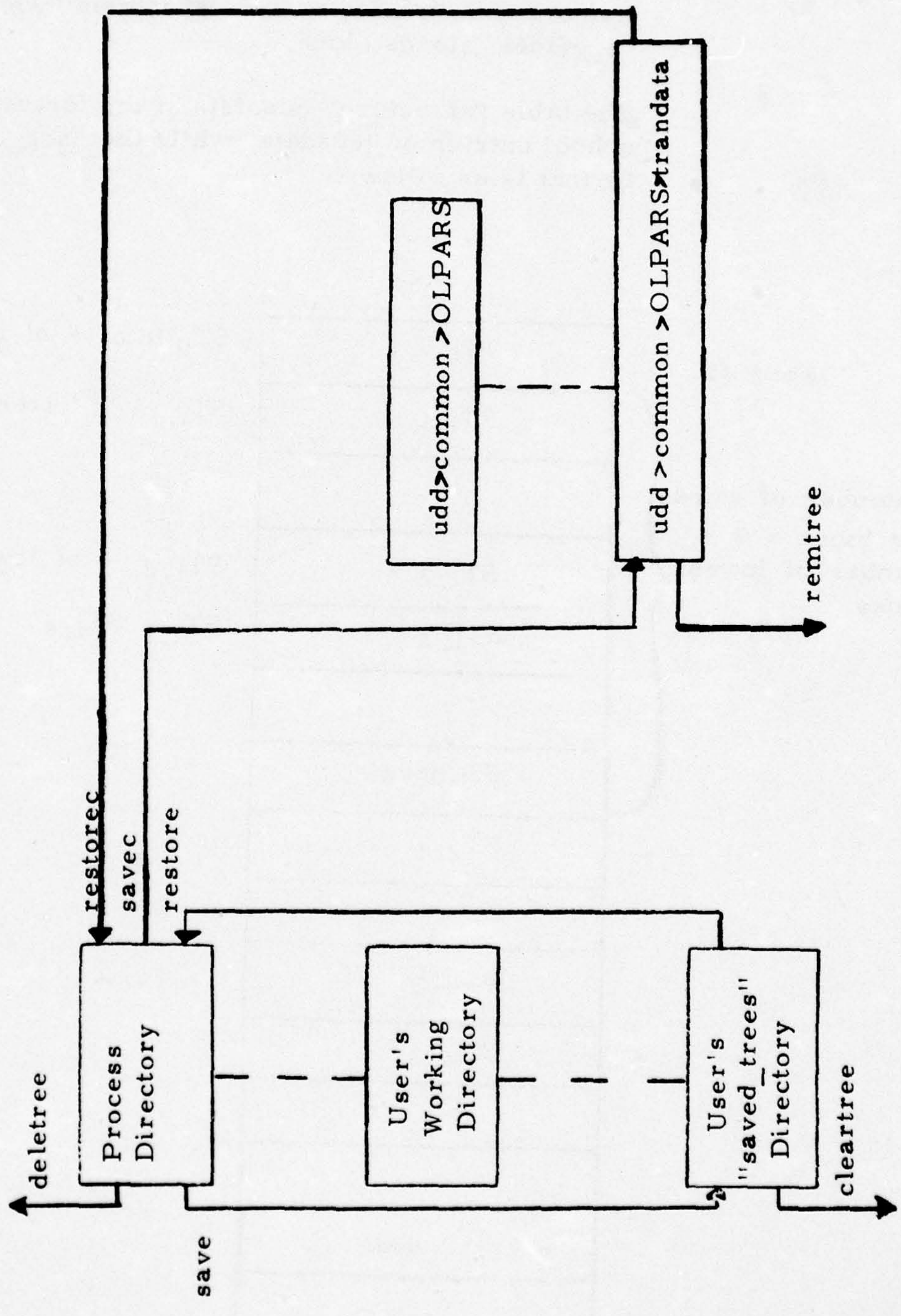
Internal Subroutine Name — s_p

Output File Settings: The specified trees are either saved, restored or deleted. In general, the different entry points are adjusting one or more of the following directories and tables:

user's "saved_trees"	accessed through "save, "restore", "cleartree"
user's "seg_o_trees"	
user's "structure"	
trandata's "saved_trees"	accessed through "savec" "restorec", and "remtree"
trandata's "seg_o_trees"	
trandata's "structure"	
"sysdata"	accessed through "deletree"

General Description: This is an overview of "s_p" with discussions of the specific entry points following. The different entry points are called by the utility functions: "save", "restore", "savec", "restorec" , "cleartree", "remtree", and "deletree". The following diagram shows the destinations of the trees involved for each specific function.

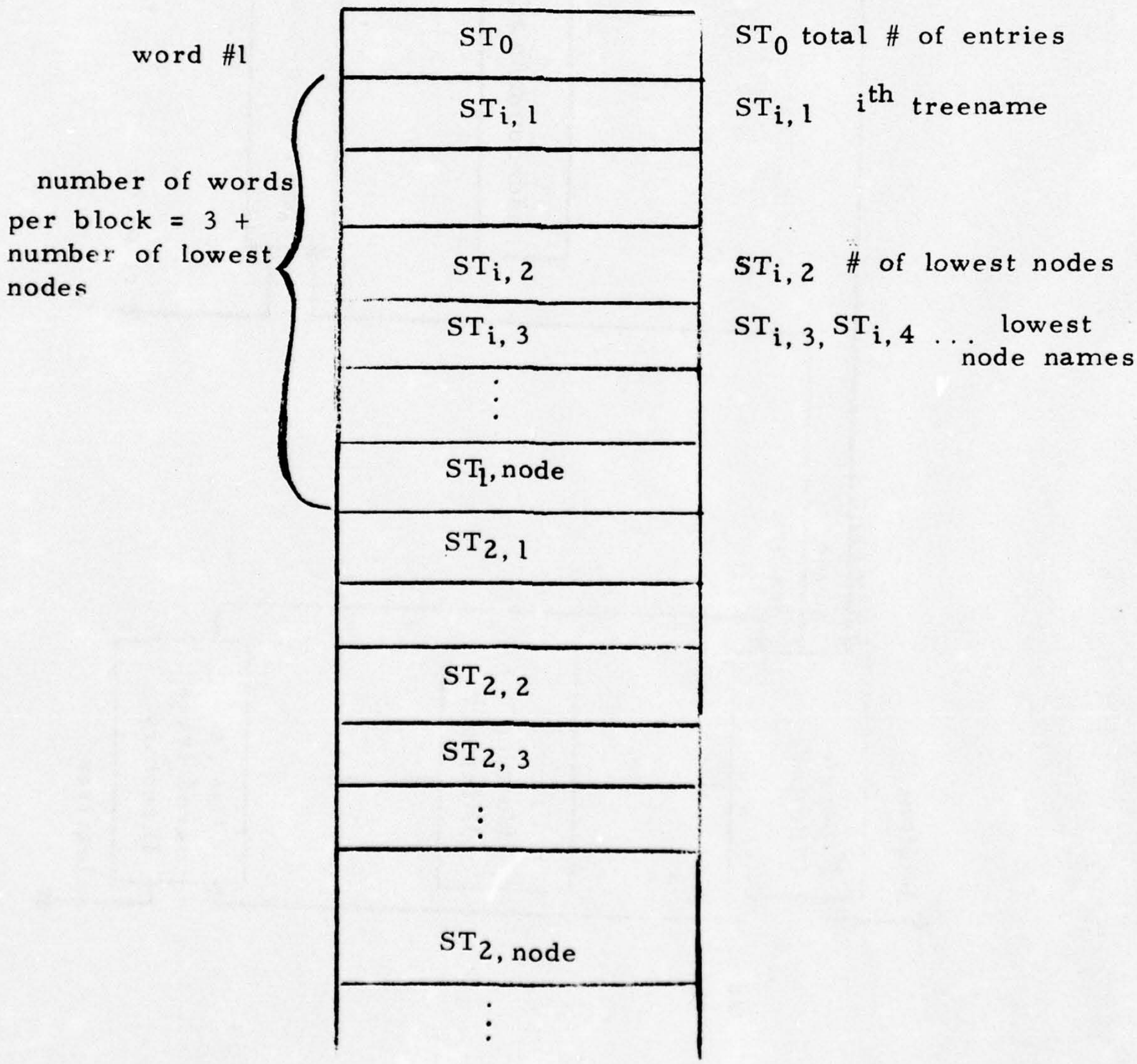
Flow Chart: See following page



udd>common>OLPARS
udd>common>OLPARS>trandata
remtree
restorec
savec
restore
Process Directory
User's Working Directory
User's "saved_trees" Directory
deletree
save
cleartree

The entry points "s_p$tsave" and "s_p$tin" are very similar, the difference is the destination of the saved tree. The entry points "s_p$trst" and "s_p$tout" also differ only in the source of the returning tree. The location of the tree to be deleted is the distinction between "s_p$tclr" and "s_p$trem" while "s_p$tdel" stands alone.

The table "structure" consists of the forest and school entries of "sysdata" while the "seg_o_trees" format is as follows:



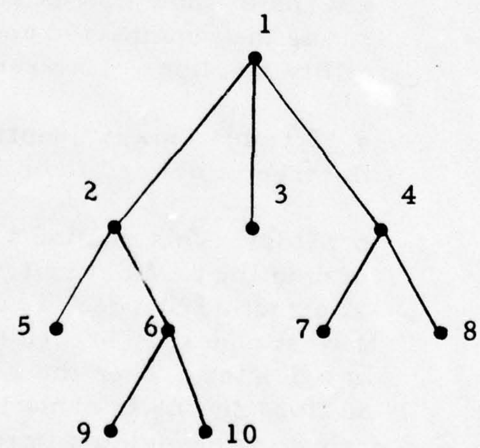
The data class names are saved as 12 character filenames (treename nodename) while the tree name files are stored with the same name or a unique name, if the real name already exists in seg_o_trees.

Program Descriptions:

s_p$tsave: This routine saves a tree in the user's "saved_trees" directory. This directory is created if it does not exist. For each tree to be saved, the directory "seg_o_trees" is examined for any occurrence of a duplicate tree name.

If one is found, an error message is printed and the user is asked to input a new name. This will be the storage name of the tree. The tree name files and data class files are then copied into saved_trees.

The structure of the tree, in "sysdata" is then copied into the "structure" file in "saved_trees" with the order of storage in the following tree diagram being: 5, 9, 10, 6, 2 , 3, 7, 8, 4, 1



Control is then returned to the calling utility function "save".

s_p$tin: The execution of the program is the same as "s_p$tsave" except the data is stored in " >udd >olpars > common>trandata" instead of the user's "saved_trees".

s_p$trst: This routine returns a tree from the user's "saved_trees" directory to the process directory. For each tree being restored, "sysdata" is examined for existing trees with a duplicate name. If this happens, the user is asked to return the tree under a unique 8 character name.

The tree names and node names are copied back into the process directory. The structure of the tree is obtained from the "structure" file and the restoring of the nodes is done in the same fashion described in "s_p$tsave".

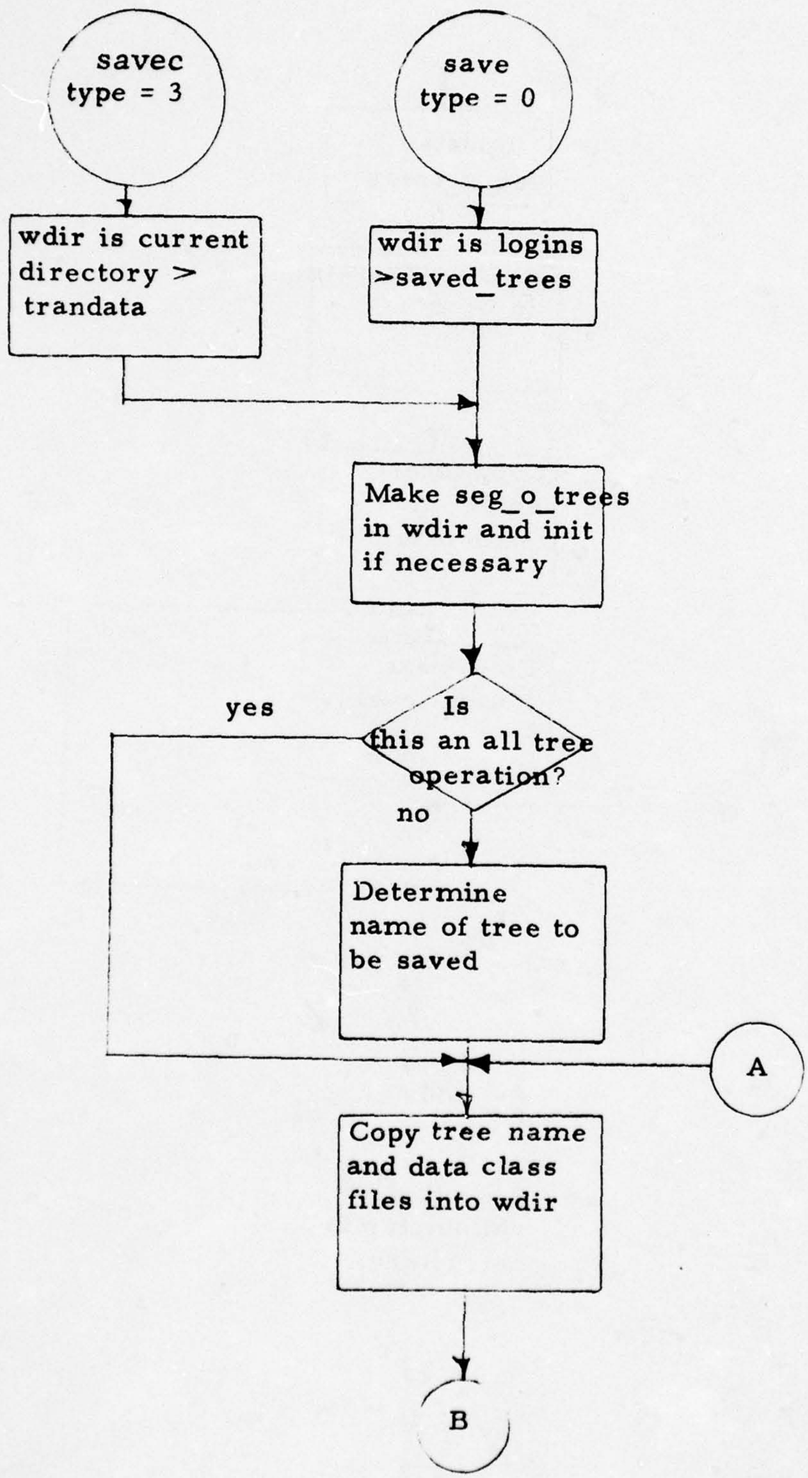
s_p$tout: The flow of the program is the same as s_p$trst" except that in "restorec", the trees are returned from "trandata", the common access directory.

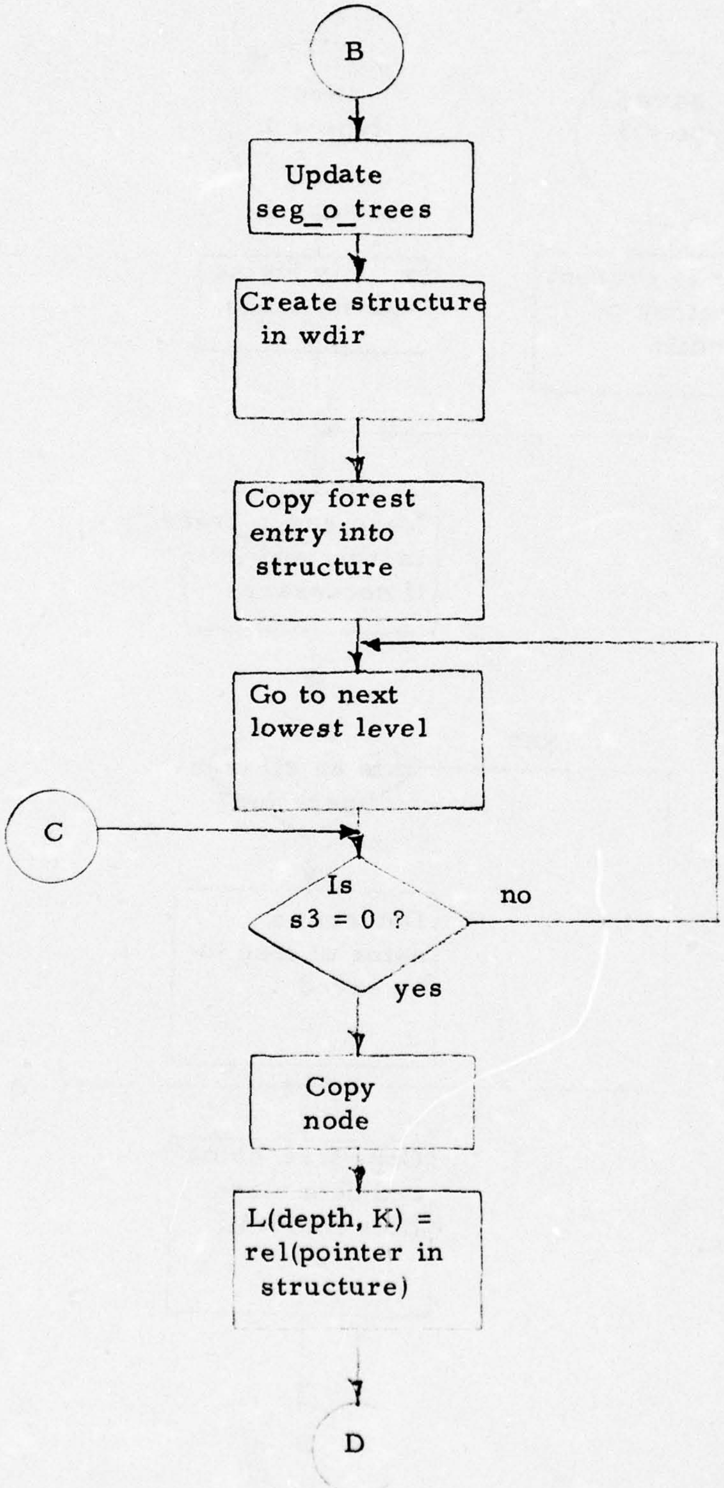
s_p$tclr: This program allows a user to delete a tree structure existing in his "saved_trees" directory. The stored tree name and data class files are removed and the "structure" file is altered to show this deletion. The "seg_o_trees" file is then compacted and control is returned to the utility function "cleartree".

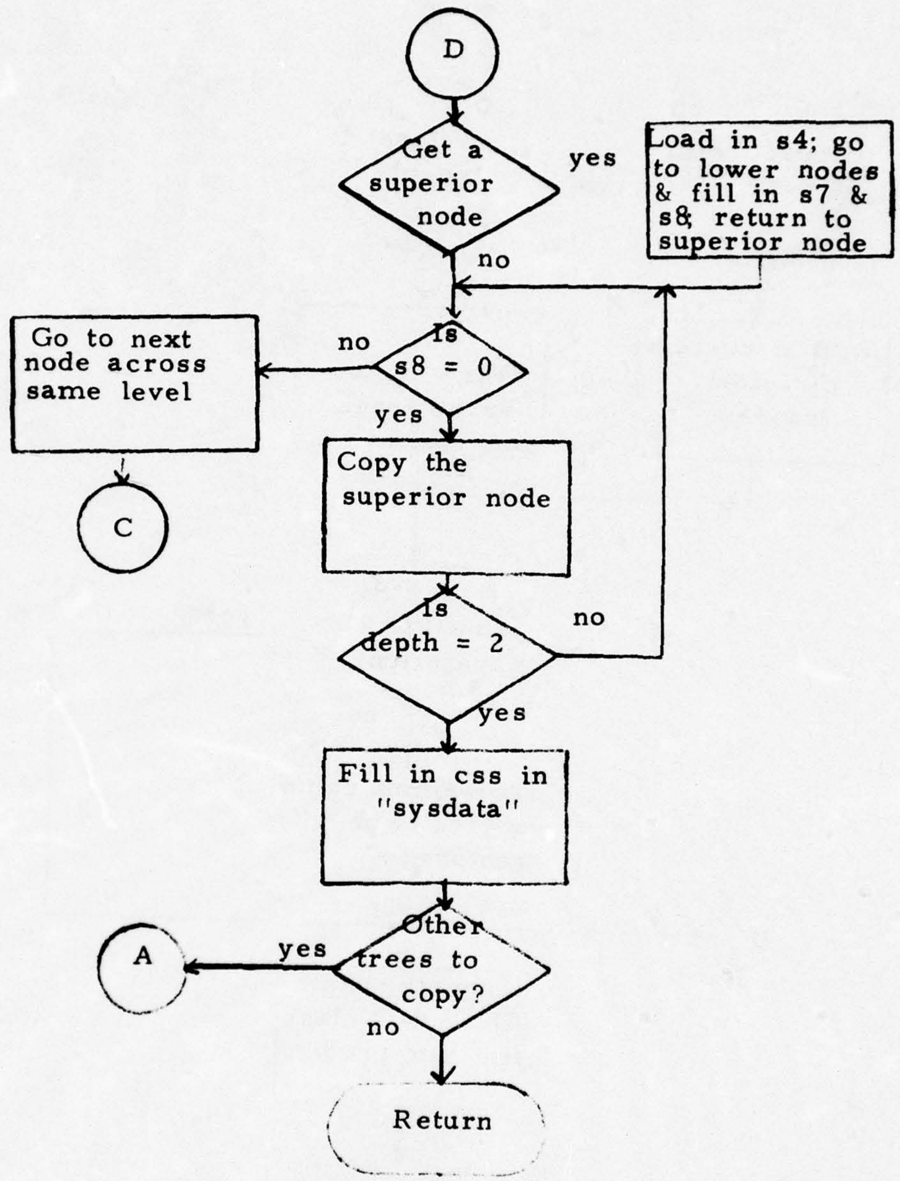
s_p$trem: This is identical to "s_p$tclr" except that the tree is deleted from the "trandata" directory.

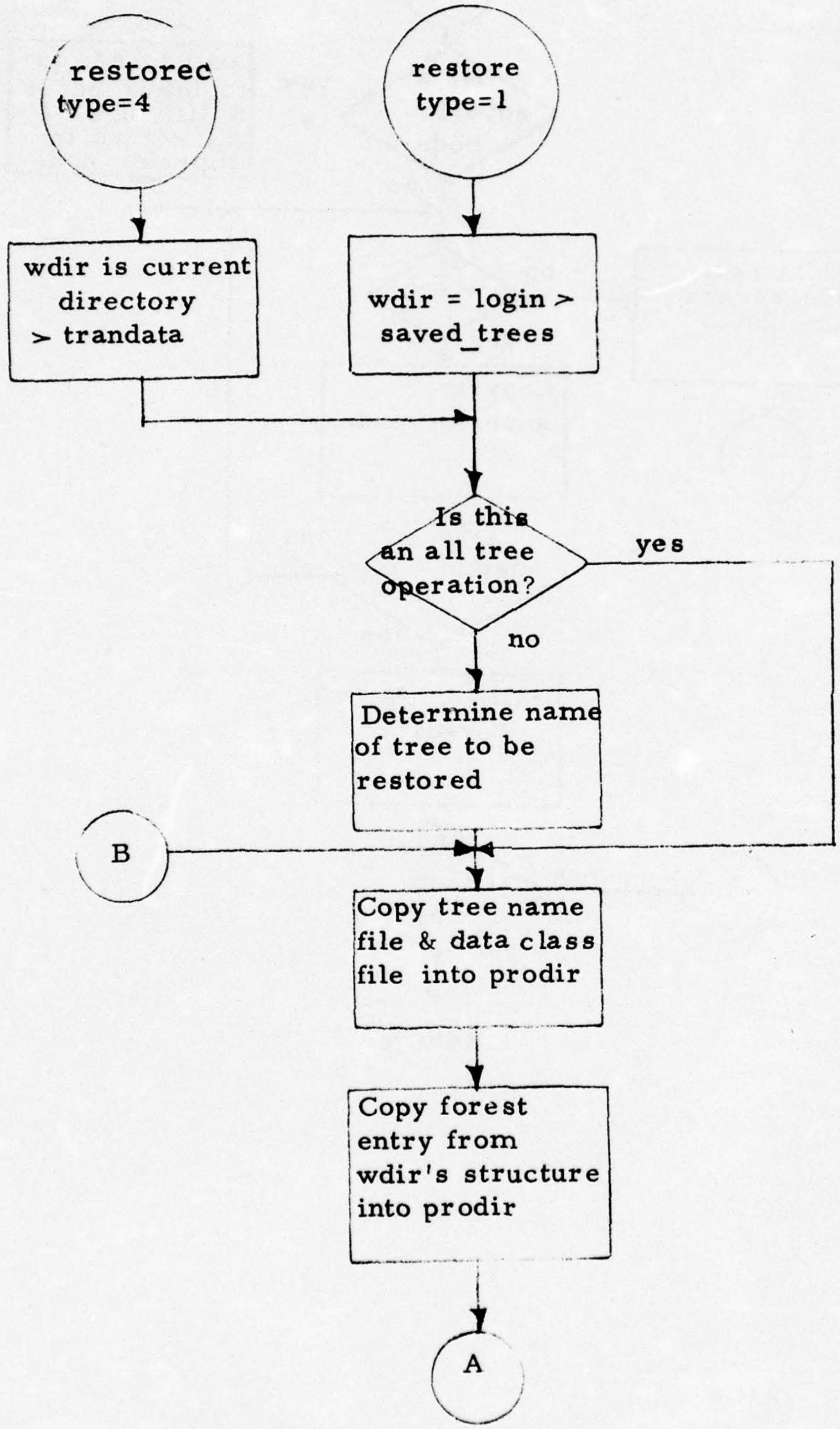
s_p$tdel: This routine allows the user the capability of deleting an existing tree in his process directory. The routine "lnodes" is used to return an array of lowest nodes. Through this list the data class files are deleted. Then the routine "getclass" is used in initializing the information of "sysdata" that pertains to the deleted tree.

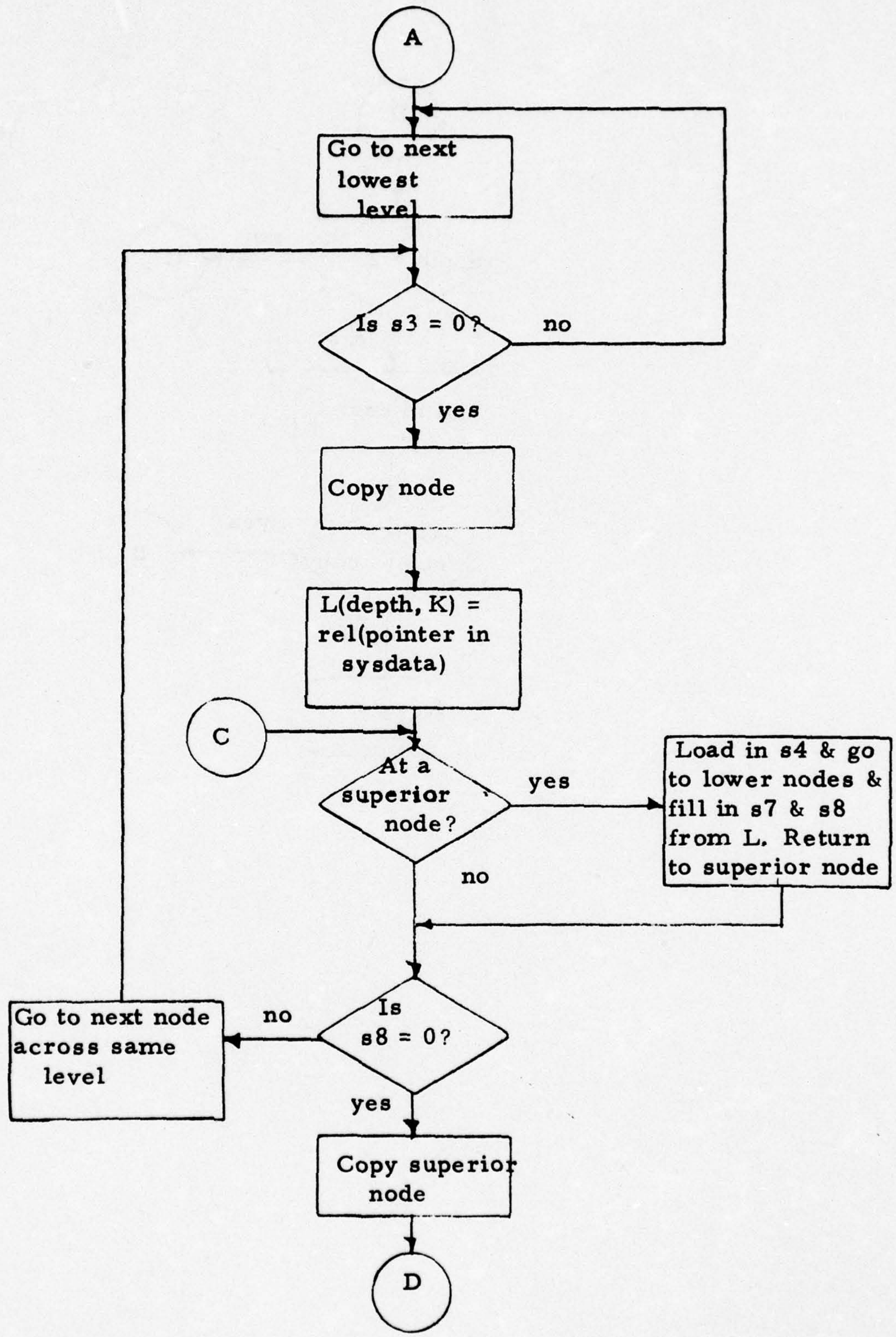
Flow Chart: See following page

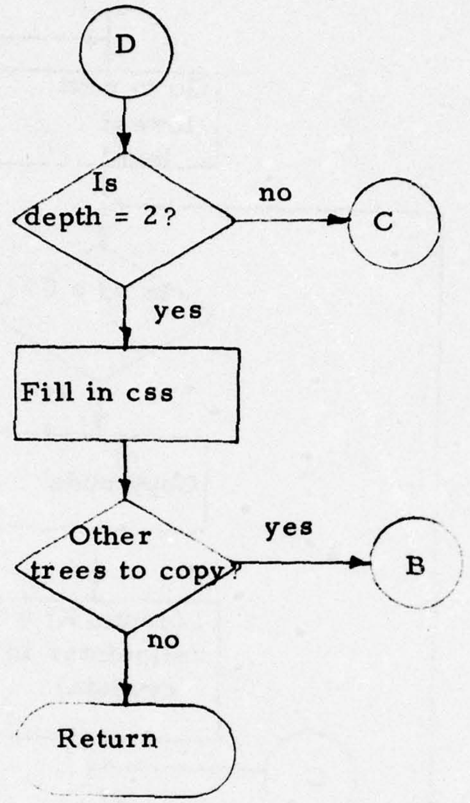

savec
type = 3
save
type = 0
wdir is current
directory >
trandata
wdir is logins
>saved_trees
Make seg_o_trees
in wdir and init
if necessary
Is
this an all tree
operation?
yes
no
Determine
name of tree to
be saved
A
Copy tree name
and data class
files into wdir
B

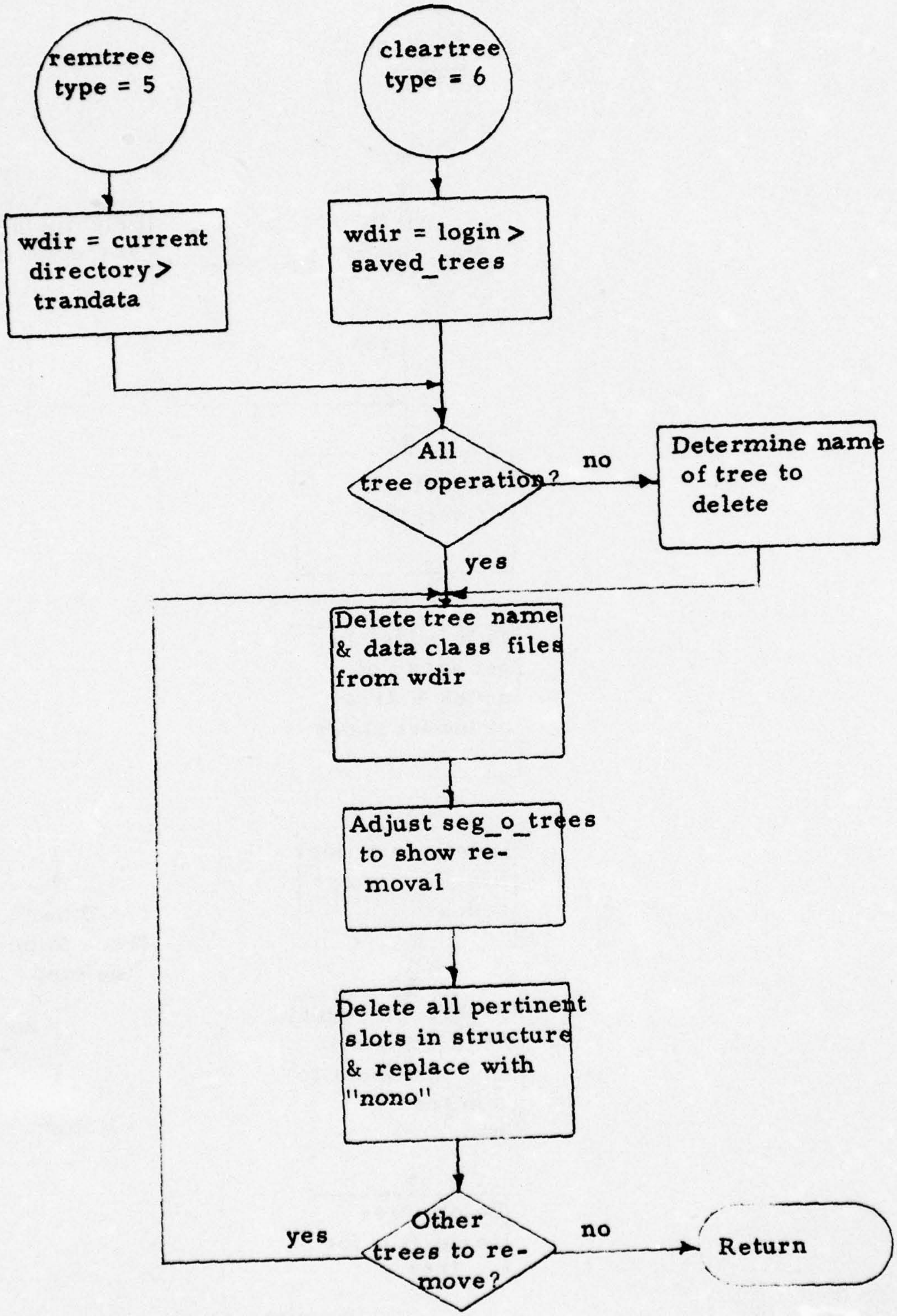

B
Update seg_o_trees
Create structure in wdir
Copy forest entry into structure
Go to next lowest level
C
Is s3 = 0 ?
no
yes
Copy node
L(depth, K) = rel(pointer in structure)
D

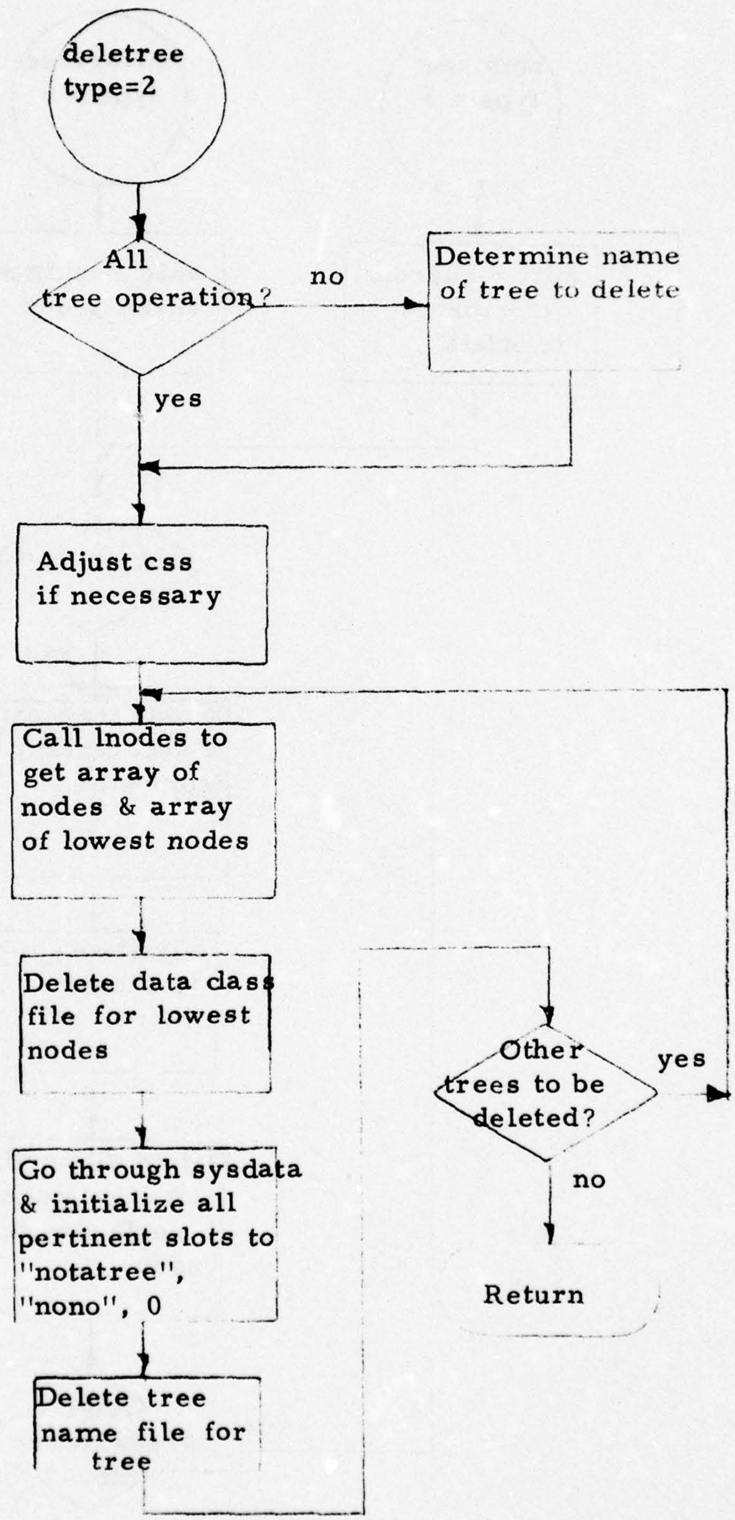

D
Get a superior node
yes
Load in s4; go to lower nodes & fill in s7 & s8; return to superior node
no
Is s8 = 0
no
Go to next node across same level
C
yes
Copy the superior node
Is depth = 2
no
yes
Fill in css in "sysdata"
Other trees to copy?
yes
A
no
Return


restorec
type=4
restore
type=1
wdir is current
directory
> trandata
wdir = login >
saved_trees
Is this
an all tree
operation?
yes
no
Determine name
of tree to be
restored
B
Copy tree name
file & data class
file into prodir
Copy forest
entry from
wdir's structure
into prodir
A


A
Go to next lowest level
Is s3 = 0?
no
yes
Copy node
L(depth, K) = rel(pointer in sysdata)
C
At a superior node?
yes
Load in s4 & go to lower nodes & fill in s7 & s8 from L. Return to superior node
no
Is s8 = 0?
no
Go to next node across same level
yes
Copy superior node
D


D
Is
depth = 2?
no
C
yes
Fill in css
Other
trees to copy?
yes
B
no
Return


remtree
type = 5
cleartree
type = 6
wdir = current directory > trandata
wdir = login > saved_trees
All tree operation?
no
Determine name of tree to delete
yes
Delete tree name & data class files from wdir
Adjust seg_o_trees to show removal
Delete all pertinent slots in structure & replace with "nono"
Other trees to remove?
yes
no
Return


deletree type=2
All tree operation?
no
Determine name of tree to delete
yes
Adjust css if necessary
Call lnodes to get array of nodes & array of lowest nodes
Delete data class file for lowest nodes
Go through sysdata & initialize all pertinent slots to "notatree", "nono", 0
Delete tree name file for tree
Other trees to be deleted?
yes
no
Return

Utility Function Name: save

Calling Sequence: type in "save(treename/"all")"

Input Parameters:

treename	specify a particular data set
"all"	perform operation for all data sets in "sysdata"

Output File Settings:

Directory	user's "saved_trees" will be expanded
Tables	user's "seg_o_trees" will be expanded
Segments	user's "structure" will be expanded

Program Description: This routine calls s_p$tsave and returns control to the user. The appropriate entries are made in "seg_o_trees" which acts as a directory and is referenced by the program "s_p". As named, the segment "structure" which is similar to "sysdata", contains the structure of the saved data set.

Utility Function Name: savec

Calling Sequence: type in "savec(treename/"all")"

Input Parameters:

treename — specify a particular data set

"all" — perform operation for all data sets in "sysdata"

Output File Settings:

tables — trandata's "seg_o_trees" will be expanded

segments — trandata's "structure" will be expanded

Program Description: This routine calls s_p$tin and returns control to the user. The appropriate entries are made in "seg_o_trees" which acts as a directory as is referenced by the program "s_p". The segment "structure", which is similar to "sysdata" is updated to reflect the new addition(s).

Utility Function Name: scale$rt

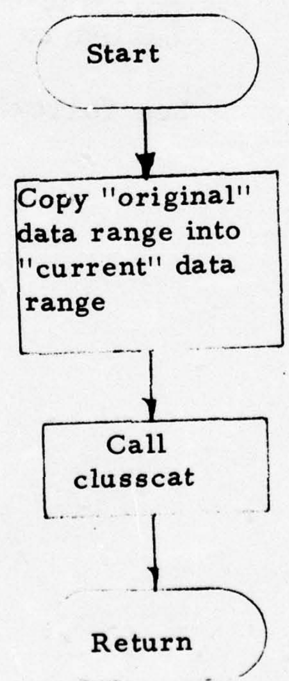
Calling Sequence: type in "scale$rt"

Output File Settings: words $D8_1$ -$D8_4$ of the two-space "display" file format are adjusted to reflect the new display.

Program Description: scale$rt copies words $D5_1$-$D5_4$ of the "display" file, the "original" data range, into words $D8_1$ -$D8_4$, the "current" data range. The program exits by calling "clusscat".

Flow Chart:

scale$rt



Utility Function Name: scale$zm

Calling Sequence: type in "scale$zm"

Output File Settings: Words D8, -$D8_4$ of the two-space "display" file format are adjusted to reflect the new bounds of the plot.

Program Description: The subroutine "multeks$read_xhair is used twice to select new lower-left and upper-right hand corners of the display. These new tektronix points are converted to data values with respect to the current display by the following formulas:

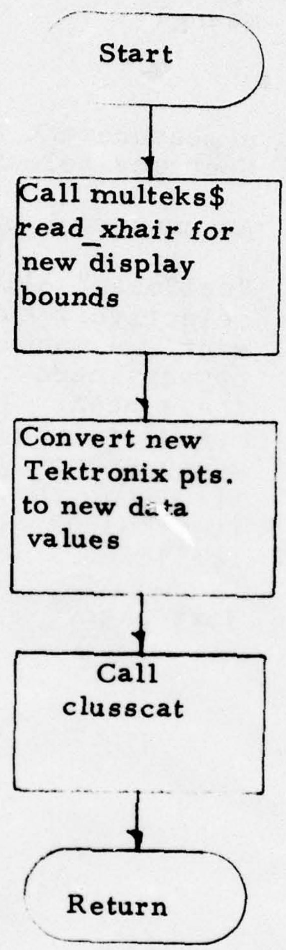
new_x = ((x-120)/724*(xmax-xmin) + xmin

new_y = ((y-53)/680)*(ymax-ymin) + ymin

The new data ranges replace the "current" data range values in the "display" file and "clusscat" is called to display the new plot.

Flow Chart: See following page

scale$zm


Start
Call multeks$ read_xhair for new display bounds
Convert new Tektronix pts. to new data values
Call clusscat
Return

Utility Function Name: sel$meas

Calling Sequence: type in "sel$meas($meas_1$, $meas_2$,...., $mean_n$)"

Input Parameters:

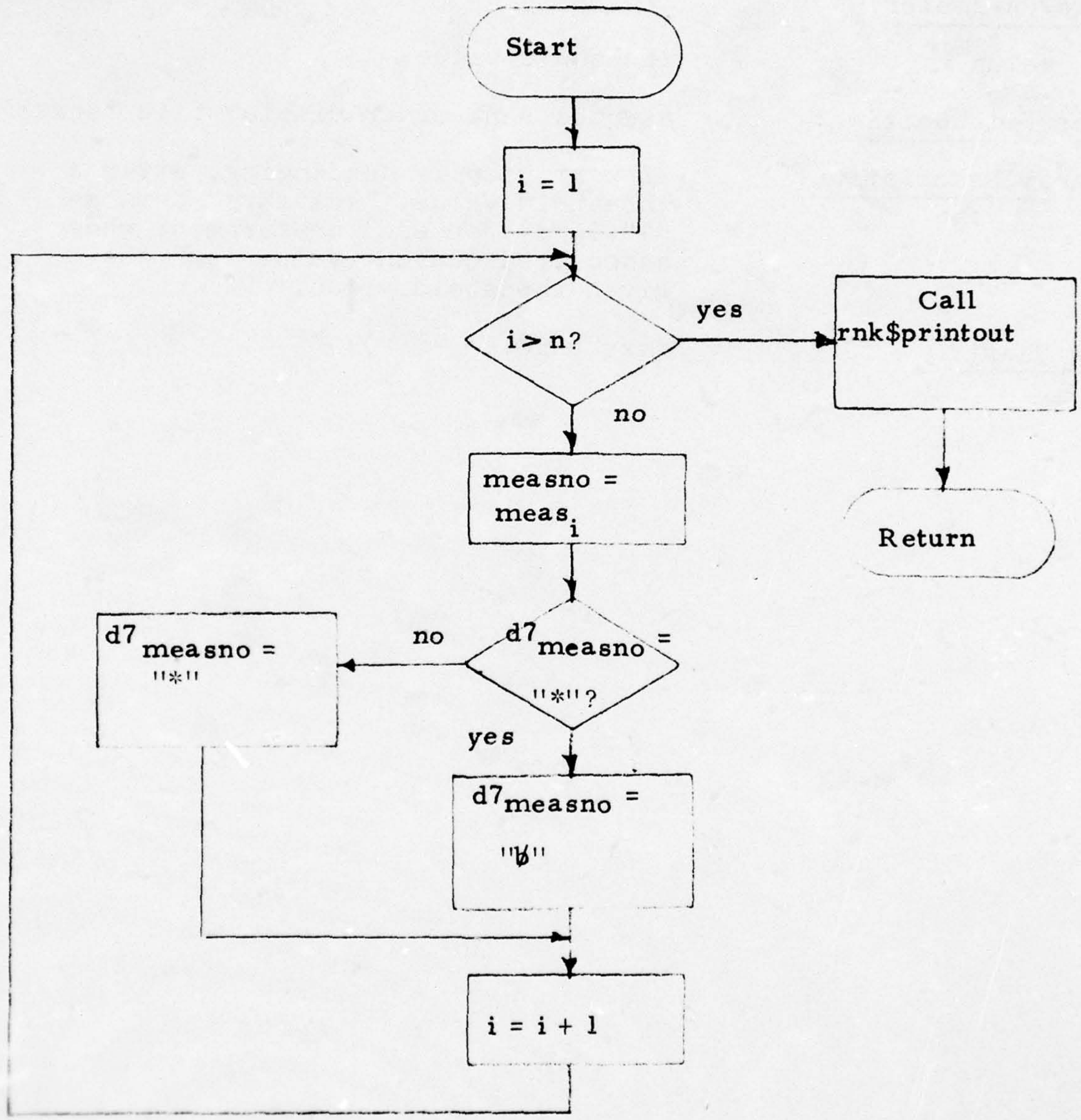
meas — a measurement number to be selected. User may select as many as he wishes.

Input File Settings: Assumes rank order display file format

Program Description: "sel$meas" allows the user to selectively turn on or off a measurement. A measurement is on if an * appears next to the measurement on the screen. The routine exclusive ors d7 of the display file for that measurement number with "*". Thus if d7 for that measurement is "*", it makes d7 blank. If blank, it makes it "*".

Flow Chart: Next page.

sel$meas


Start
i = 1
i > n?
yes
Call
rnk$printout
Return
no
measno =
meas$_i$
d7 measno =
"*"?
no
d7 measno =
"*"
yes
d7 measno =
"ƀ"
i = i + 1

Utility Function Name: sel$thrs

Calling Sequence: type in "sel$thrs (value)"

Input Parameter:

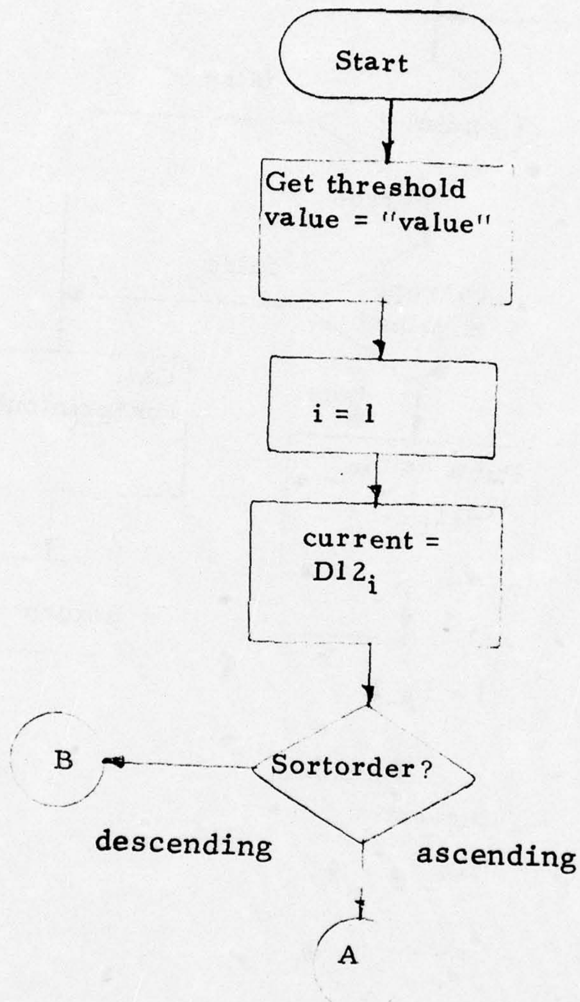
value - threshold value

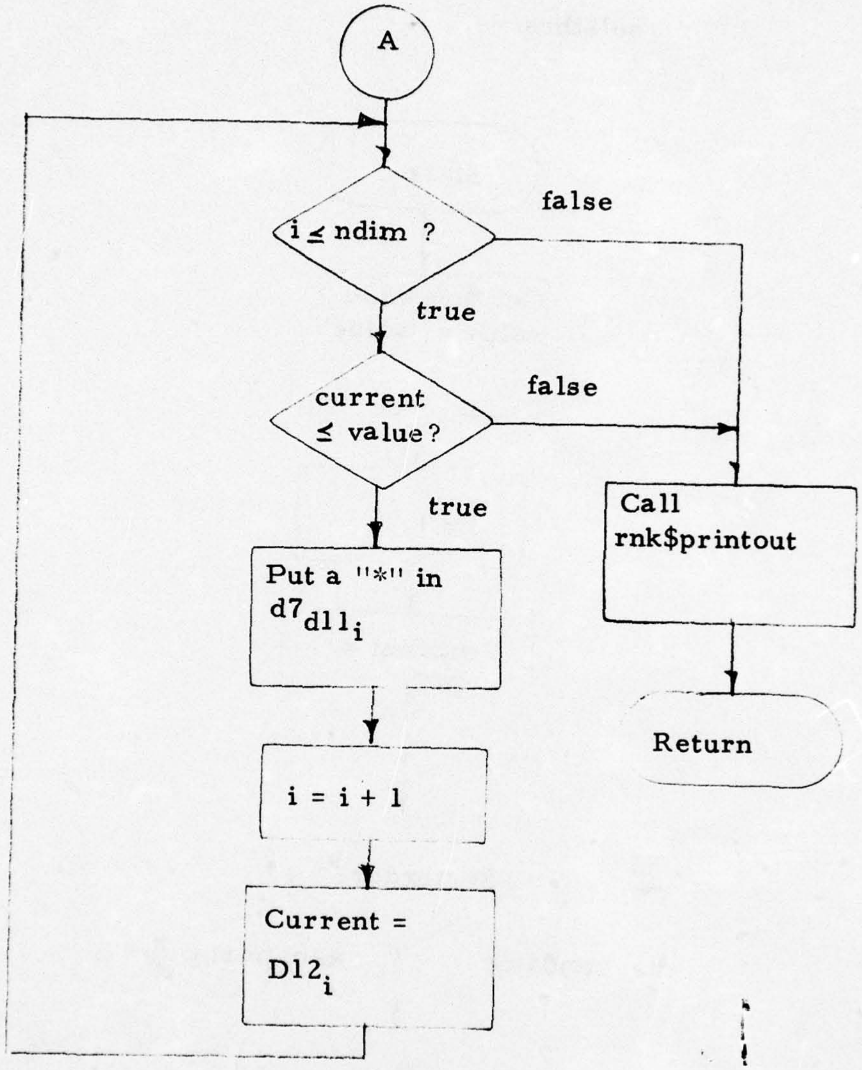
Input File Setting: assumes rank order display file format

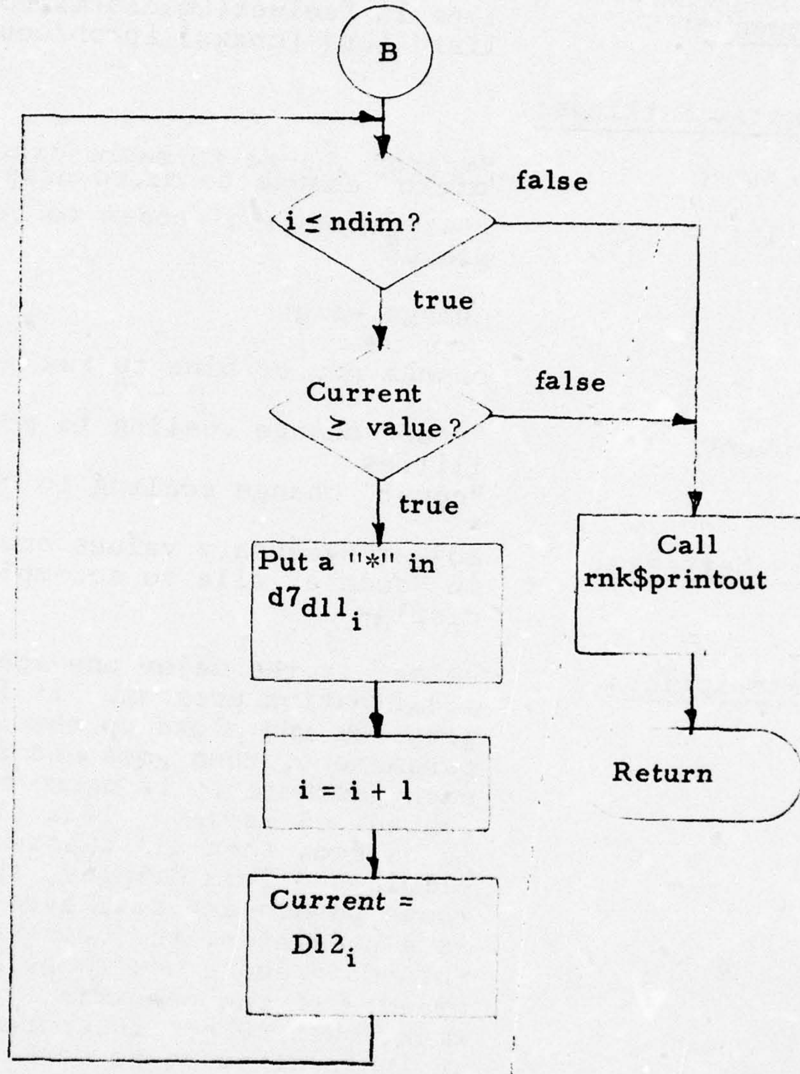
Program Description: If sortorder is descending, given a threshold value, "sel$thrs" puts an "*" in d7 for each measurement whose associated current value is $\geq$ the given threshold value.

Flow Chart: next page

sel$thrs


Start
Get threshold value = "value"
i = 1
current = D12i
Sortorder?
B
descending
ascending
A


A
i ≤ ndim ?
false
true
current
≤ value ?
false
true
Put a "*" in
d7dll_i
Call
rnk$printout
Return
i = i + 1
Current =
D12_i


B
false
i ≤ ndim?
true
false
Current ≥ value?
true
Put a "*" in $d7dll_i$
$i = i + 1$
Current = $D12_i$
Call rnk$printout
Return

User Utility Function: select

Calling Sequence: type in "select([macro/micro] [class-list] [cr] [cbxxx] [prob/count])"

Input Parameter Settings:

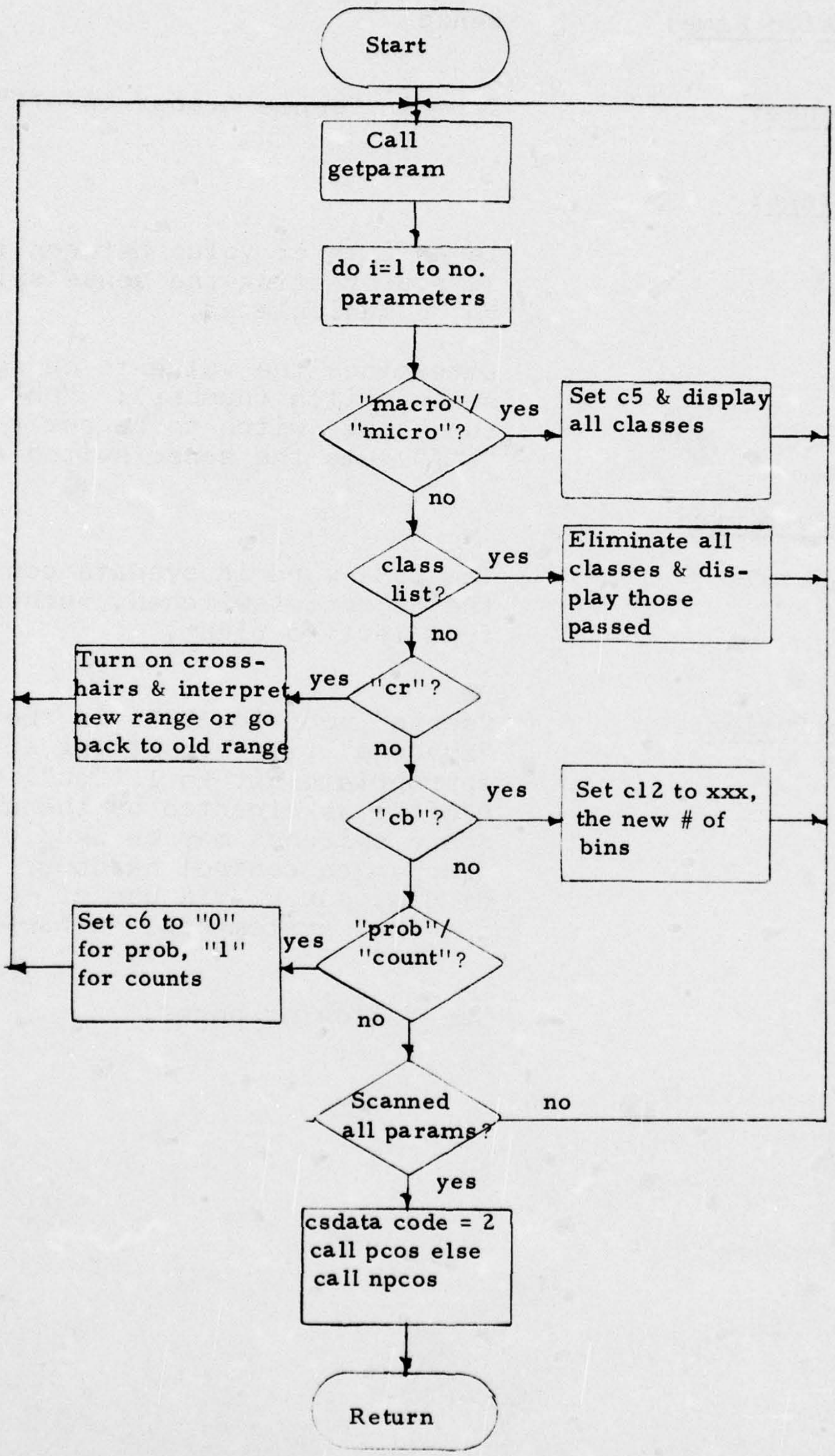
macro/micro	"macro" change to macro display "micro" change to micro display
classlist	class symbols of nodes to be displayed
cr	change range
cbxxx	change no. of bins to xxx
prob/count	"prob" change scaling to probabilities "count" change scaling to counts

Output File Settings: adjust necessary values and flags in "csdata" file to accomplish new display

Program Description: select is the major one-space display modification program. It first calls getparam and picks up the input parameters, then goes and interprets each parameter. If macro/micro is passed, c5 is set. If a class list is passed, then all classes are eliminated from display, then only those passed are displayed. If 'cr' is a parameter, the crosshairs are turned on and a new range is determined. If the new xmin > the new xmax, then select interprets this as an instruction to go back to original xmin and xmax. If cbxxx is a parameter then c12 is set to xxx. If 'prob'/'count' is passed then c6 is set accordingly. The program exits by calling "pcos" if the current moos function is probconf or else calls npcos.

Flow Chart: See following page

select


Start
Call getparam
do i=1 to no. parameters
"macro"/ "micro"?
yes
Set c5 & display all classes
no
class list?
yes
Eliminate all classes & display those passed
no
"cr"?
yes
Turn on cross-hairs & interpret new range or go back to old range
no
"cb"?
yes
Set c12 to xxx, the new # of bins
no
"prob"/ "count"?
yes
Set c6 to "0" for prob, "1" for counts
no
Scanned all params?
no
yes
csdata code = 2 call pcos else call npcos
Return

Utility Function Name: sense

Calling Sequence: Type in "sense number on/off"

Input Parameters:

(number) — Is an integer value between 1 and 36 representing the sense switch to be manipulated.

("on"/"off") — Determines the value to be set into sense switch (number): "on" causes the sense switch to be set = 1, "off" sets the sense switch = 0.

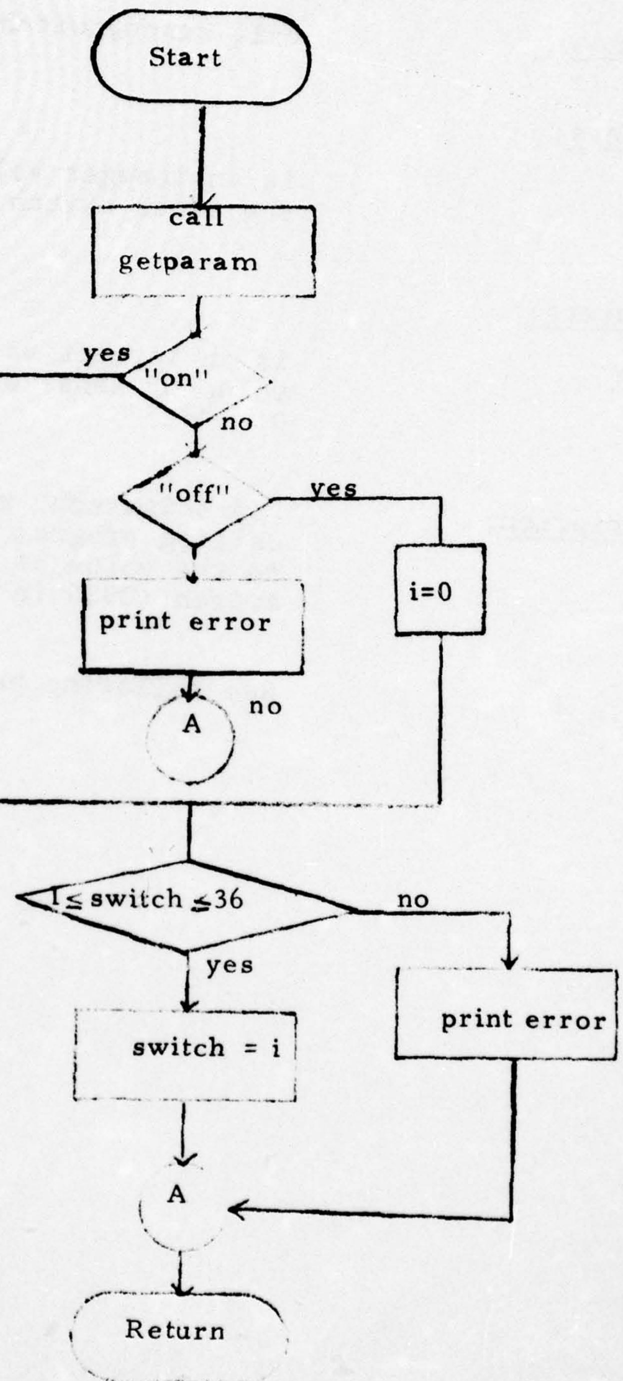
Output File Settings:

"sysdata" file — The CSS6 word in sysdata contains the 36 sense switches, numbered from left to right.

Program Description: "sense" modifies CSS6 in the "sysdata" file by setting an appropriate bit to 1 ("on") or 0 ("off") as directed by the user. Sense switches may be used by the program to control hardcopy or display output via use of the internal system call "sense$switch".

Flow Chart: See following pages.

sense


Start
call
getparam
yes
"on"
no
i=1
"off"
yes
print error
i=0
A
no
1≤ switch ≤36
no
yes
switch = i
print error
A
Return

Internal Subroutine Name: sense$switch

Calling Sequence: call sense$switch (sw,index)

Input Parameters:

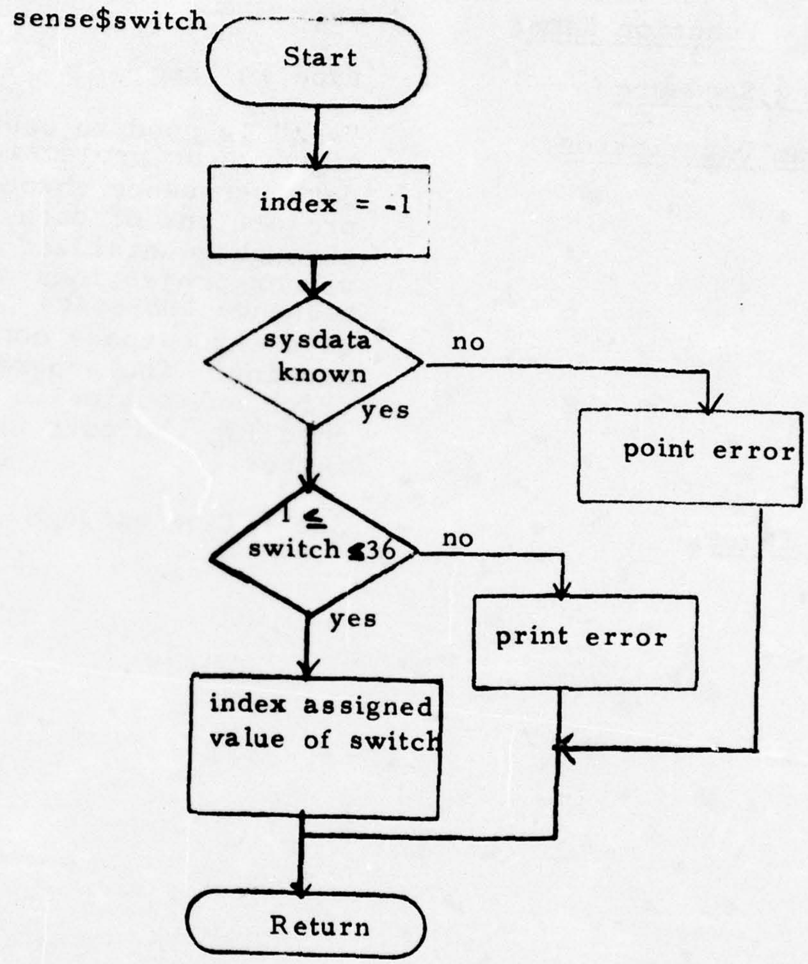
sw is an integer value representing the sense switch to be tested.

Output Parameters:

index is an integer value equal to the value of sense switch sw (1 = "on", 0 = "off").

Program Description: "sense$switch" returns to the calling program a parameter equal to the value of a specified sense switch (CSS6 in file "sysdata").

Flow Chart: See following page.



sense$switch
Start
index = -1
sysdata known
no
yes
point error
1 ≤ switch ≤36
no
yes
print error
index assigned value of switch
Return

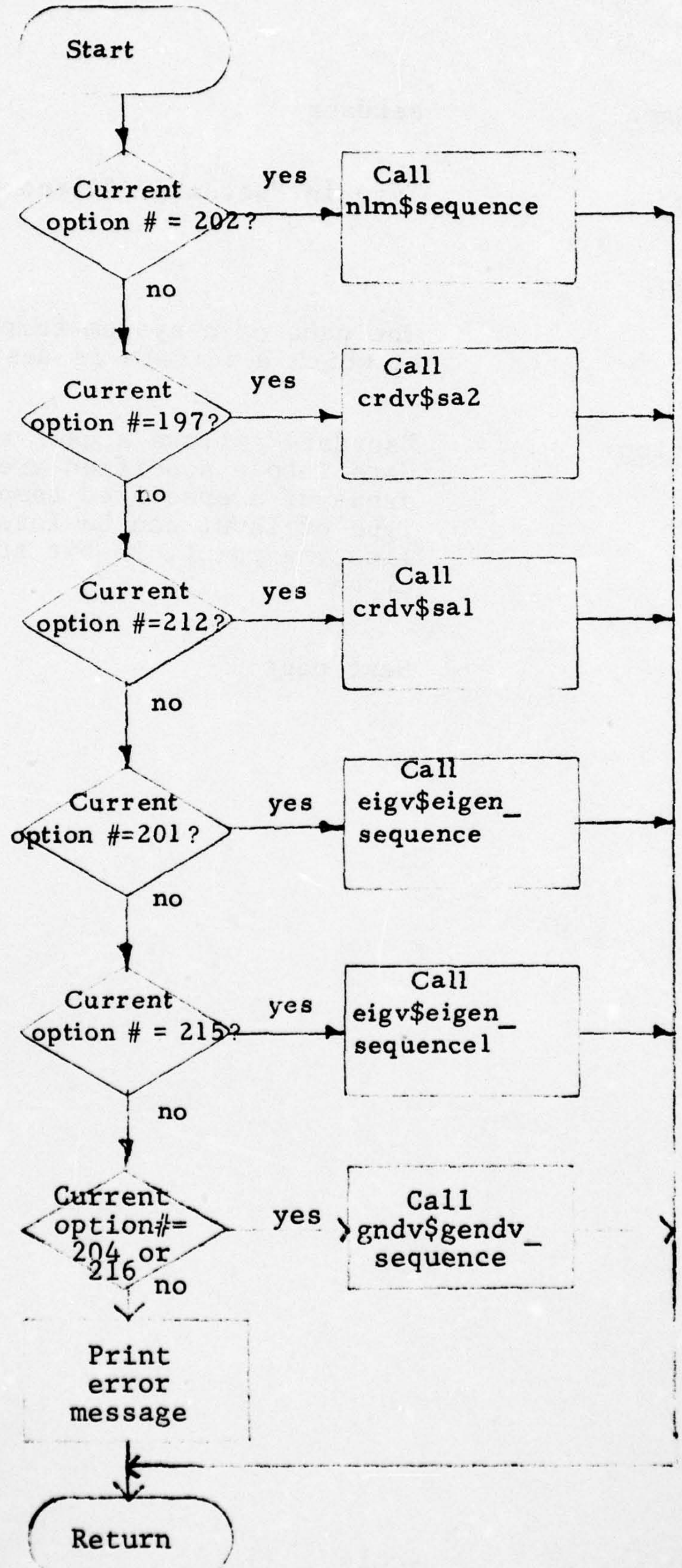
Utility Function Name: seq

Calling Sequence: type in "seq"

Program Description: "seq" is used to sequence through eigenvector projections of data, sequence through coordinate projections of data, sequence through generalized discriminant vector projections of data, and sequence two-space projections of a three-space non-linear mapping. The program determines which subroutine to call by checking the current option number.

Flow Chart: See following page.

seq


Start
Current option # = 202?
yes
Call nlm$sequence
no
Current option #=197?
yes
Call crdv$sa2
no
Current option #=212?
yes
Call crdv$sa1
no
Current option #=201?
yes
Call eigv$eigen_ sequence
no
Current option # = 215?
yes
Call eigv$eigen_ sequence1
no
Current option#= 204 or 216
yes
Call gndv$gendv_ sequence
no
Print error message
Return

Programmer Aid Name: setdata

Calling Sequence: Type in "setdata (filename)"

Input parameters:

(filename) The name of a system temporary file of which a setdata is desired.

Program Description: "setdata" allows a user to insert data into a specified area or areas of a specified temporary file. Type of input can be integer, floating point, 36-bit stream, and alpha.

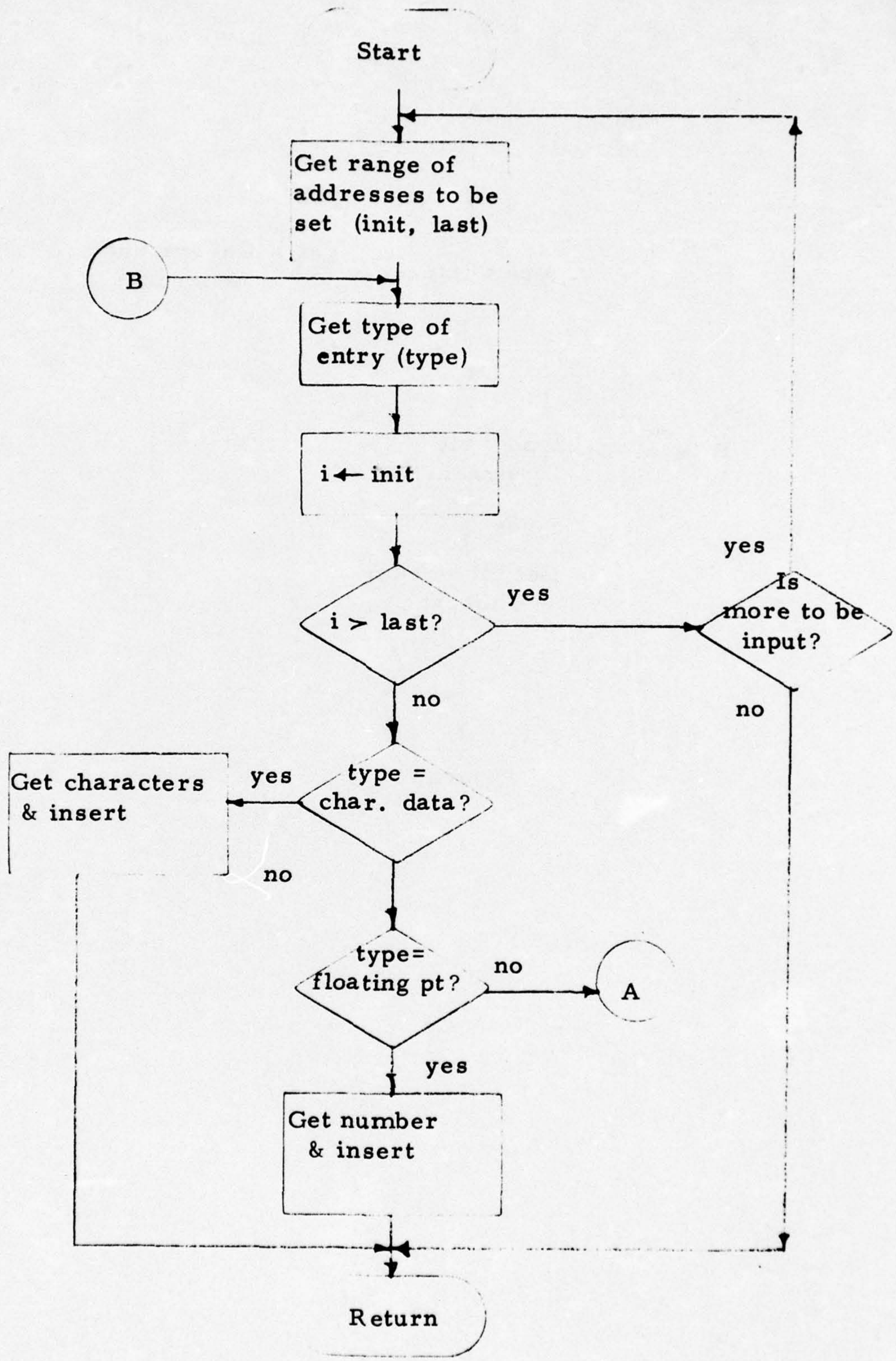
Flow Chart: Next page.

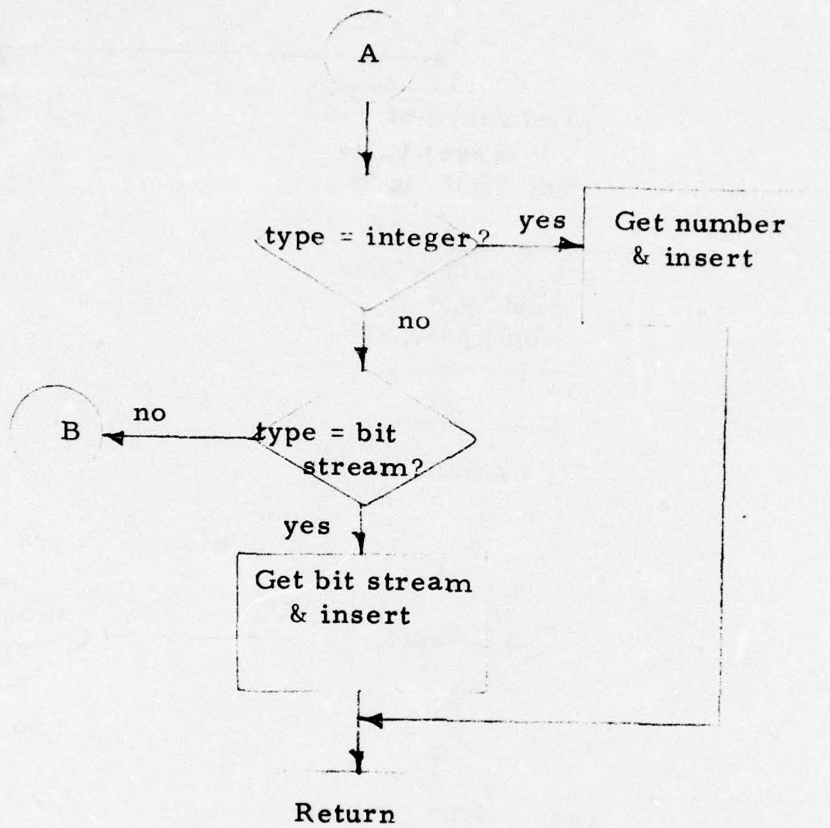
AD-A033 437 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
MULTICS OLPARS OPERATING SYSTEM.(U)
SEP 76 D B CONNELL, K N KLINGBAIL F30602-75-C-0226
UNCLASSIFIED PAR-74-25-B RADC-TR-76-271-VOL-2 NL

7 OF 7
AD
A033437

END
DATE
FILMED
2-77

setdata


Start
Get range of addresses to be set (init, last)
B
Get type of entry (type)
i ← init
i > last?
yes
Is more to be input?
yes
no
no
Get characters & insert
yes
type = char. data?
no
type= floating pt?
no
A
yes
Get number & insert
Return


A
type = integer?
yes
Get number
& insert
no
B
no
type = bit
stream?
yes
Get bit stream
& insert
Return

Internal Subroutine Name: sln

Calling Sequence: call sln (ptrs)

Input Parameters:

ptrs(1)	sysdata ptr	[dcl ptrs(5) ptr]
ptrs(2)	scratch ptr	
ptrs(3)	display ptr	
ptrs(4)	tree name file ptr	

Output Parameter:

ptrs(5) ptr to logic file

Output File Settings:

scratch

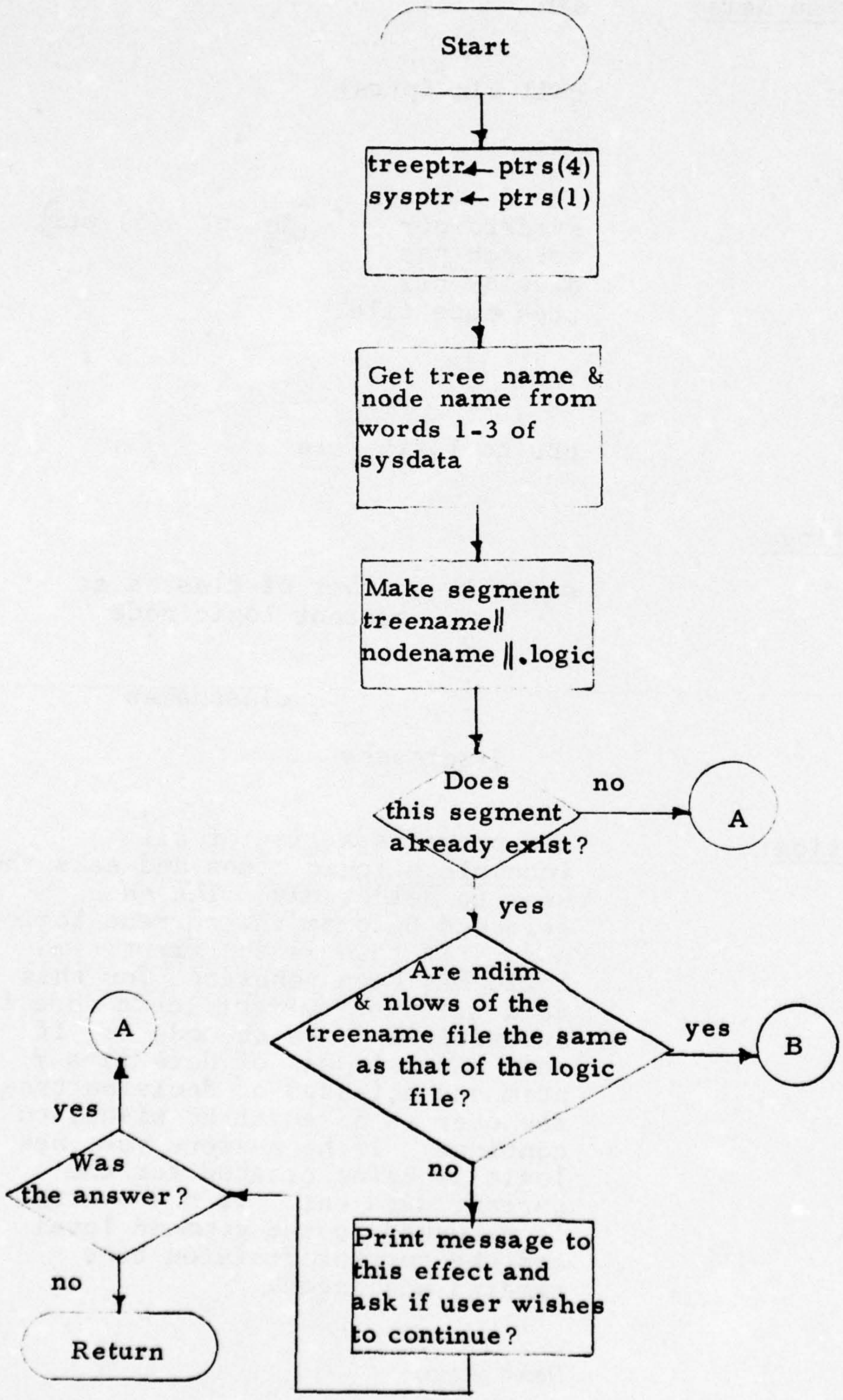
word 1 - number of classes at current logic node
2
.
. classnames
.
1+nclasses

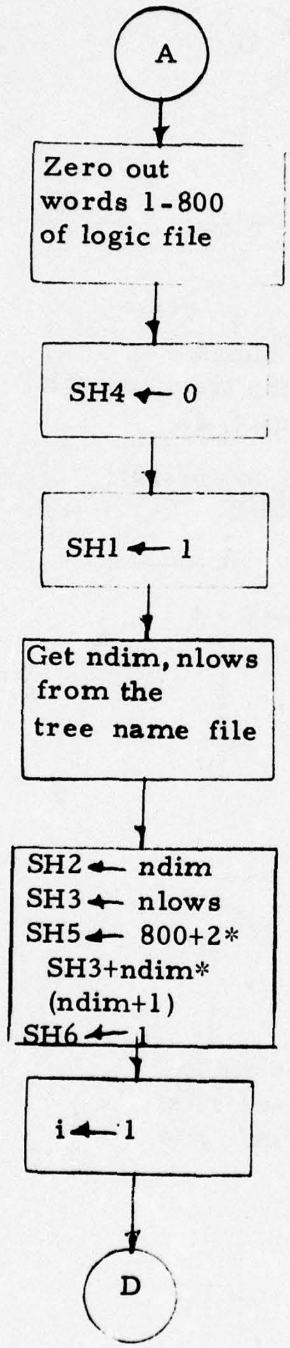
Program Description:

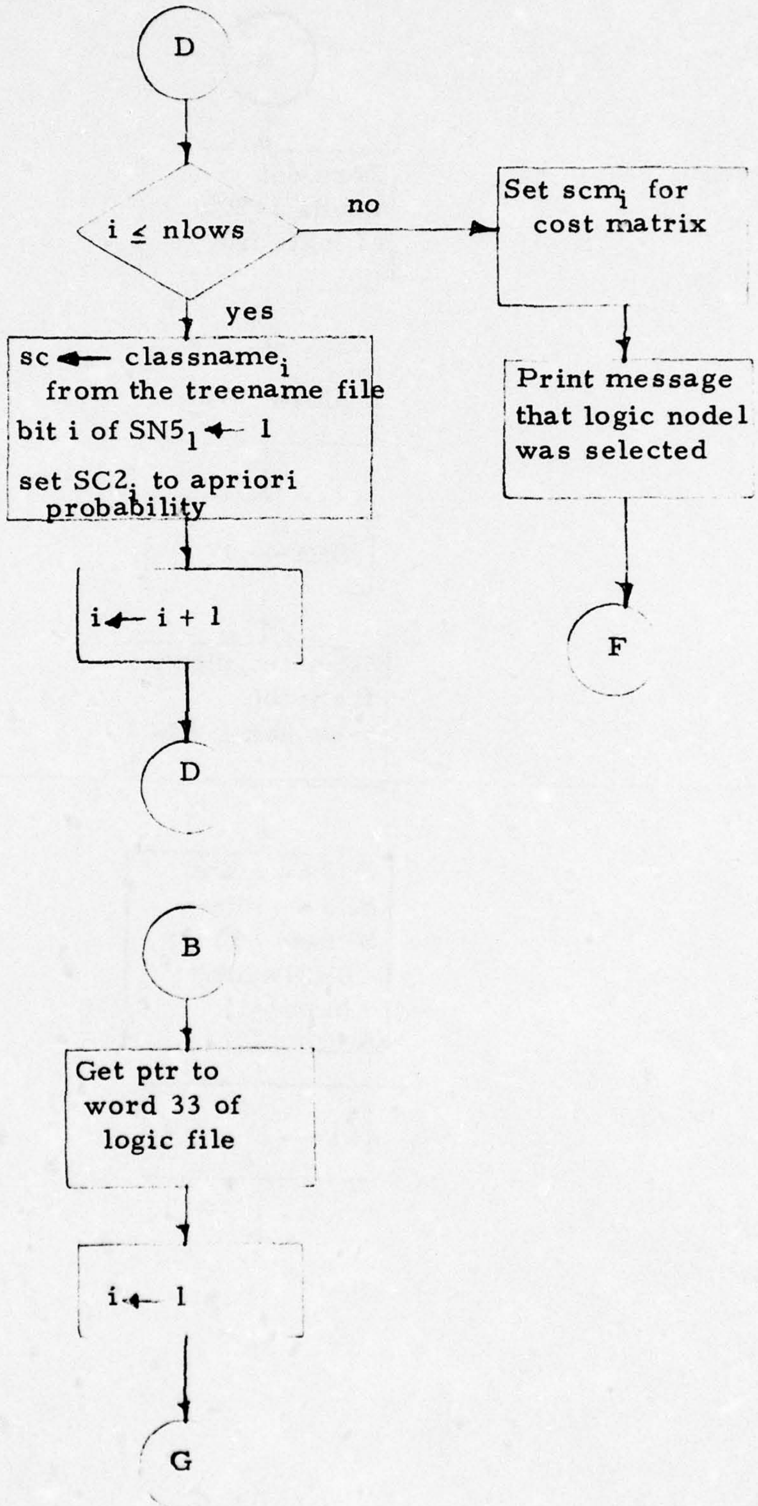
sln presents a list of all incomplete logic nodes and asks the user to select one. The one selected becomes the current logic node. If this is the first time logic has been generated for this data set, the current logic node is automatically set to node 1: If ndim and nclasses of data tree $\neq$ ndim and nclasses of decision tree, the user is asked if he wishes to continue. If he answers yes, new logic is being created for the current data set. If no, control is returned to the command level and the current decision tree remains unchanged.

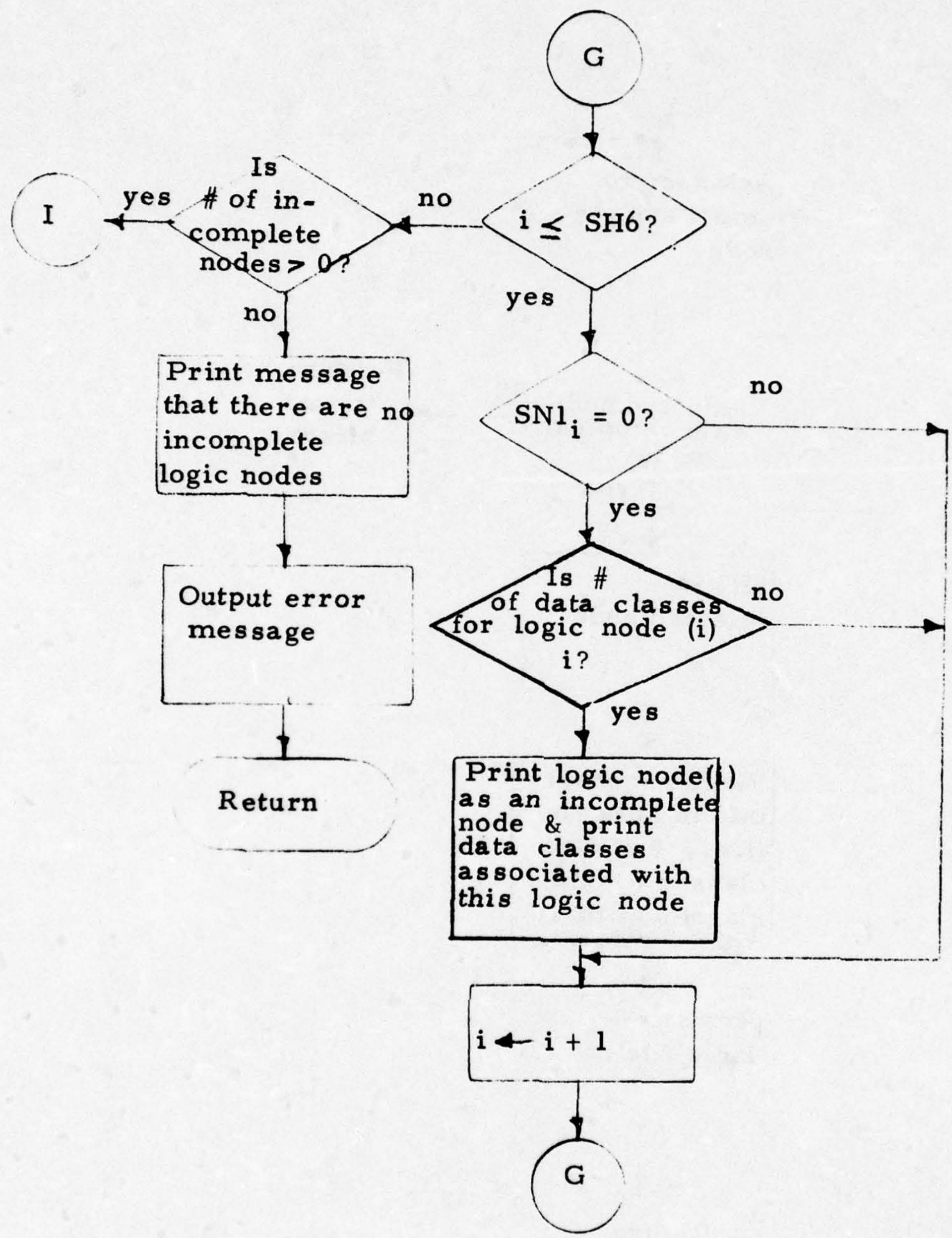
Flow Chart: Next Page

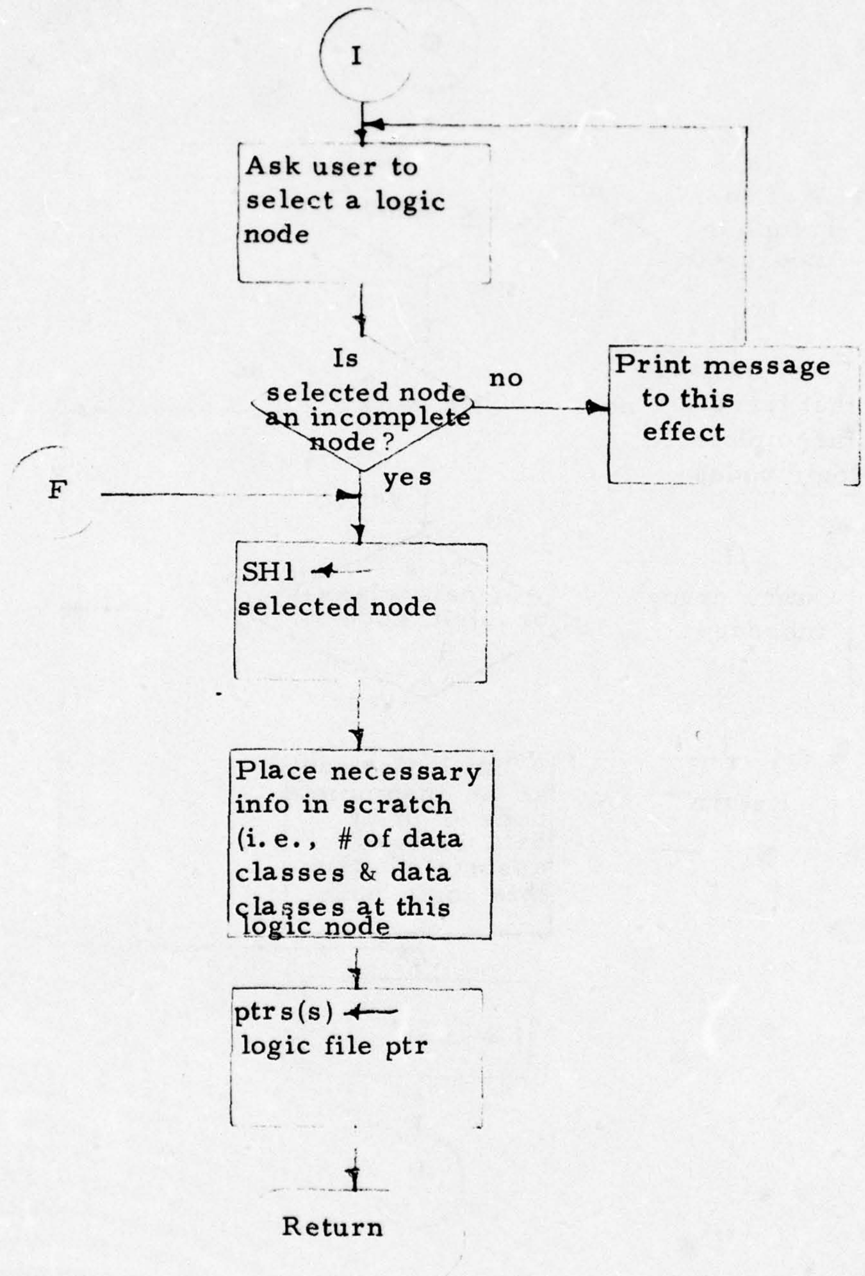
sln


Start
treeptr ← ptrs(4)
sysptr ← ptrs(1)
Get tree name & node name from words 1-3 of sysdata
Make segment treename‖ nodename ‖.logic
Does this segment already exist?
no
A
yes
Are ndim & nlows of the treename file the same as that of the logic file?
yes
B
no
Print message to this effect and ask if user wishes to continue?
Was the answer?
yes
A
no
Return


A
Zero out words 1-800 of logic file
SH4 ← 0
SH1 ← 1
Get ndim, nlows from the tree name file
SH2 ← ndim
SH3 ← nlows
SH5 ← 800+2* SH3+ndim* (ndim+1)
SH6 ← 1
i ← 1
D


D
i ≤ nlows
no
Set scm_i for cost matrix
yes
sc ← $classname_i$ from the treename file
bit i of $SN5_1$ ← 1
set $SC2_i$ to apriori probability
i ← i + 1
D
Print message that logic nodel was selected
F
B
Get ptr to word 33 of logic file
i ← 1
G


G
i ≤ SH6 ?
no
yes
Is # of in-complete nodes > 0?
yes
I
no
Print message that there are no incomplete logic nodes
Output error message
Return
SN1$_i$ = 0 ?
no
yes
Is # of data classes for logic node (i) i?
no
yes
Print logic node(i) as an incomplete node & print data classes associated with this logic node
i ← i + 1
G


I
Ask user to select a logic node
Is selected node an incomplete node?
no
Print message to this effect
yes
F
SH1 ← selected node
Place necessary info in scratch (i.e., # of data classes & data classes at this logic node
ptrs(s) ← logic file ptr
Return

Internal Subroutine Name: sln$cp

Calling Sequence: "call sln$cp (ptrs, flag)";

Input Parameters:

ptrs(1)	-	sysdata ptr
ptrs(2)	-	scratch ptr
ptrs(3)	-	display ptr
ptrs(4)	-	treename file ptr

Output Parameters:

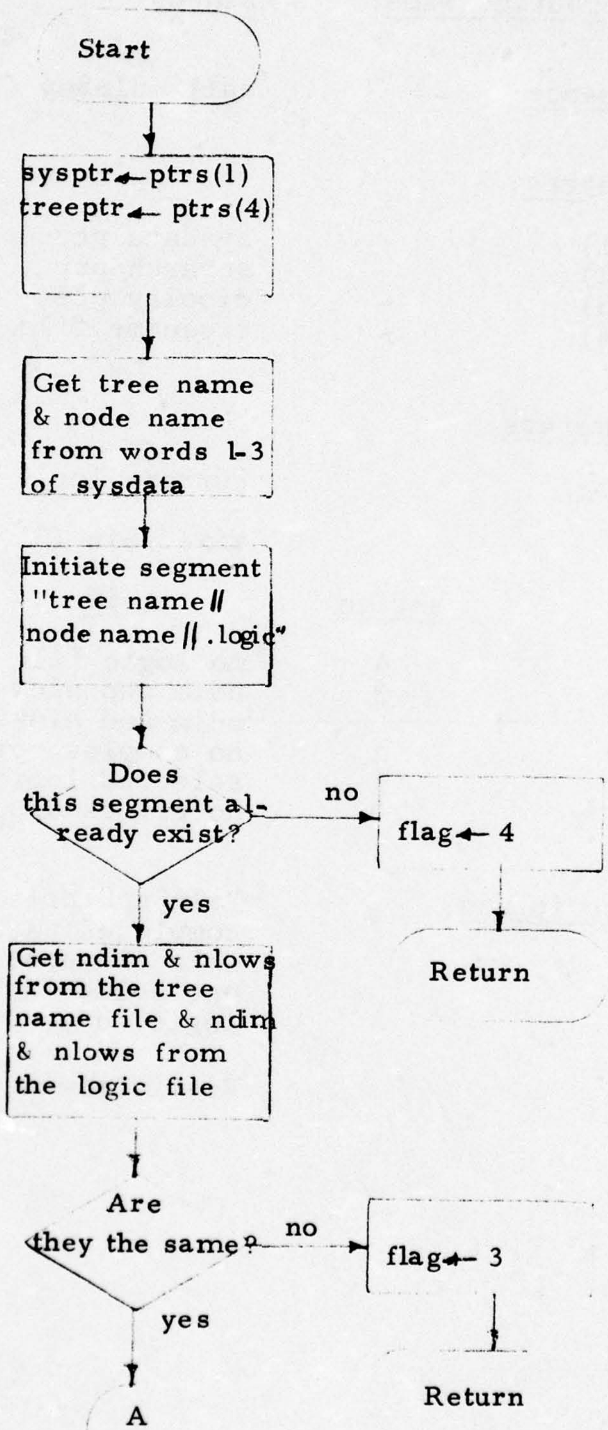
ptrs(5)	-	current logic file ptrs
flag		fixed bin (35)

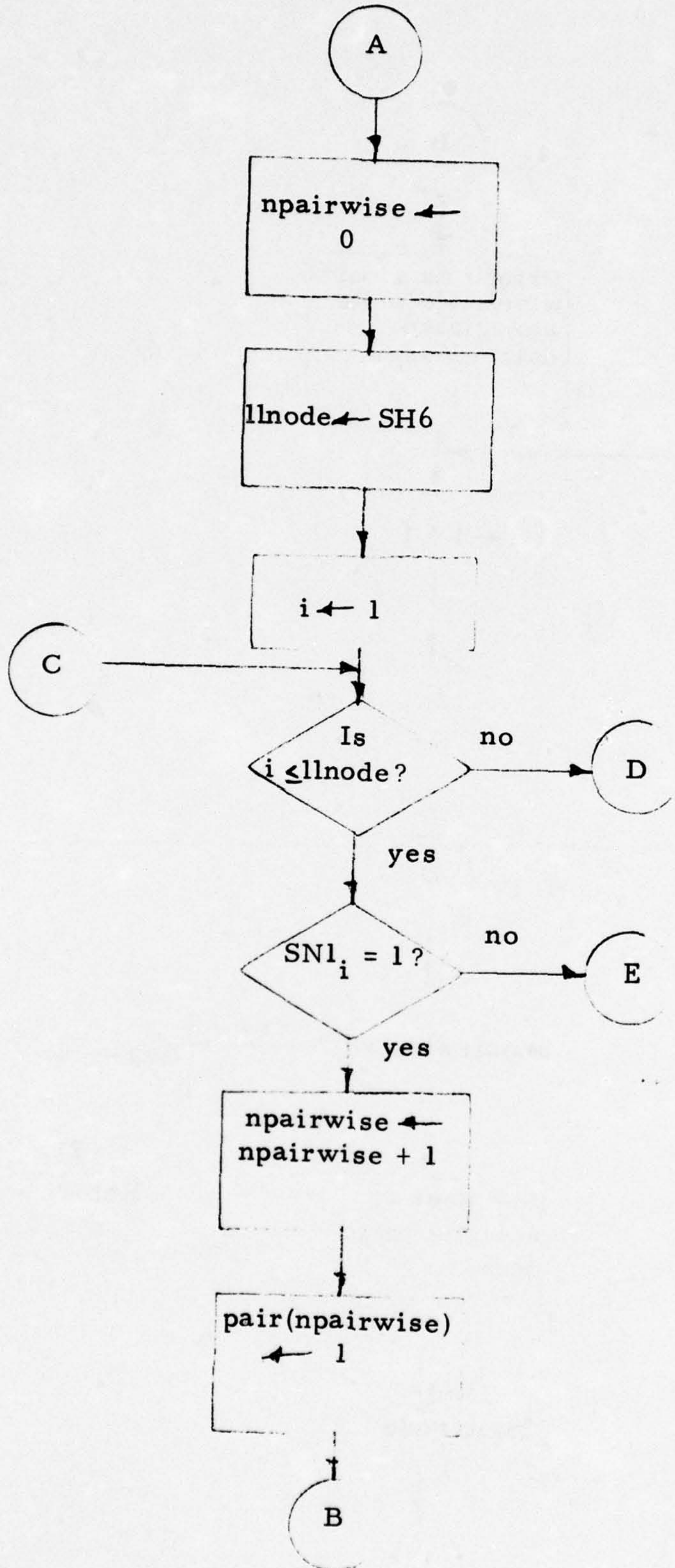
set to	if
4	no logic file
3	ndim and nlows of treename file ≠ ndim and nlows of logic file
2	no complete pairwise nodes exist
1	selected logic node is illegal
0	no errors occur

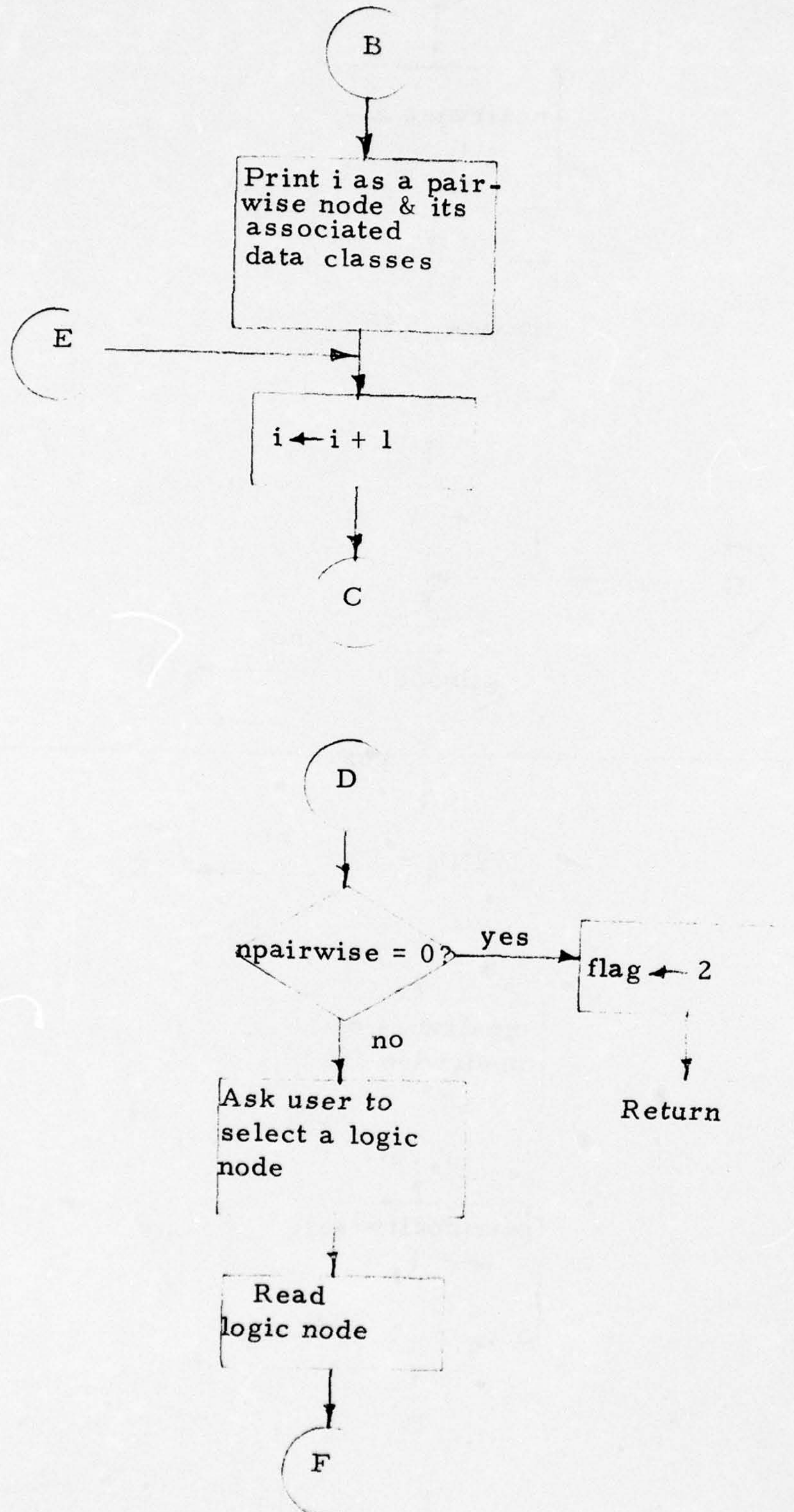
Program Description: "sln$cp" presents a list of completed pairwise logic nodes and asks the user to select one. The one selected becomes the current logic node.

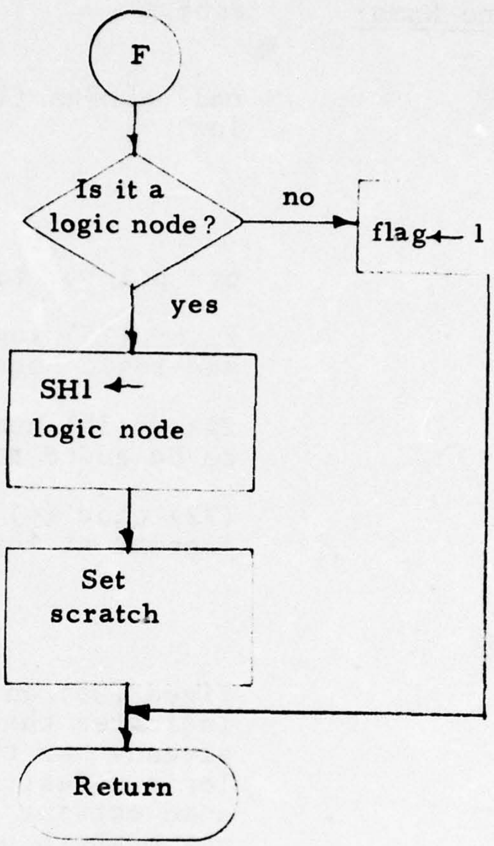
Flow Chart: See following page

sln$cp


Start
sysptr ← ptrs(1)
treeptr ← ptrs(4)
Get tree name & node name from words 1-3 of sysdata
Initiate segment "tree name // node name // .logic"
Does this segment already exist?
no
flag ← 4
Return
yes
Get ndim & nlows from the tree name file & ndim & nlows from the logic file
Are they the same?
no
flag ← 3
Return
yes
A


A
npairwise ← 0
llnode ← SH6
i ← 1
C
Is i ≤ llnode?
no
D
yes
SN1_i = 1?
no
E
yes
npairwise ← npairwise + 1
pair(npairwise) ← 1
B


B
Print i as a pair-wise node & its associated data classes
E
i ← i + 1
C
D
npairwise = 0?
yes
flag ← 2
Return
no
Ask user to select a logic node
Read logic node
F


F
Is it a logic node ?
no
flag ← 1
yes
SH1 ← logic node
Set scratch
Return

Internal Subroutine Name: sln$gn

Calling Sequence: call sln$gn (lptr, sn, nn, ncls, low)

Input Parameters:

lptr — ptr pointer to the mooslogic file

sn — fixed (35) logic node under which new logic nodes are being added

ncls — fixed (35) number of logic nodes to be added to MOOSLOGIC file

low — (72) char (4) array of node names present at logic node sn.

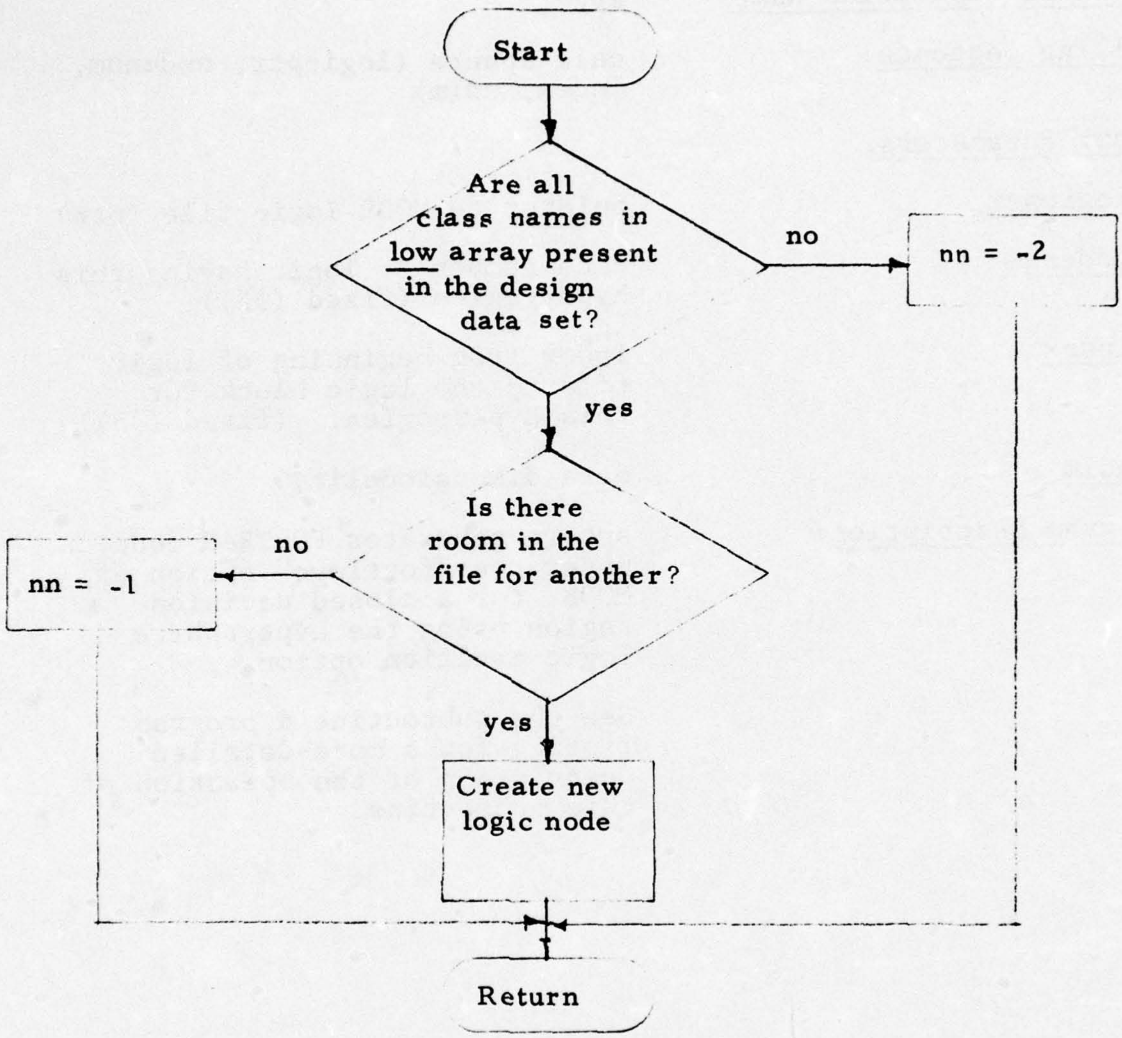
Output Parameters:

nn — fixed (35) an error flag. -1 indicates that the mooslogic file already has the maximum number of logic nodes. -2 indicates that some entries in the low array are not members of the data set on which logic is being designed.

Program Description: sln$gn finds a free slot for a new logic node in the node part of the structure part of the mooslogic file. If a slot can be found, it creates a new logic node using the information that was passed to it and returns. If any element in the array of class names does not correspond to the data set on which logic is being designed, nn is set to -2. If the node part of the structure part of the mooslogic file is full, nn is set to -1.

Flow Chart: See following page

sln$gn


Start
Are all class names in low array present in the design data set?
no
nn = -2
yes
Is there room in the file for another?
no
nn = -1
yes
Create new logic node
Return

Internal Subroutine Name: sphere

Calling Sequence: call sphere (logicptr, nodenum, index, ndim)

Input Parameters:

logicptr	pointer to MOOS logic file (ptr)
nodenum	node number of logic having this hyperregion (fixed (35))
index	index from beginning of logic file to the logic block for this hyperregion. (fixed (35))
ndim	data dimensionality

Program Description: sphere generates FORTRAN code, under the "fortlogc" option of MOOS, for a closed decision region using the hypersphere logic creation option

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Internal Subroutine Name: ss$all_part

Calling Sequence: call ss$all_part(i)

Output Parameter:

i — fixed (35) set to 1 if only vectors which fall at a node are to be used, 0 other wise (see program description).

Program Description: ss$all_part asks the question: "Should the covariance matrix and mean vectors used in calculations be computed from only those vectors which fall at a given logic node?"

If the user answers yes, "i" is set to 1, otherwise, "i" is set to 0.

Internal Subroutine Name: ss$display

Calling Sequence: call ss$display (ndim, nn, ptrs, low, vl, v2, tc, x)"

Input Parameters:

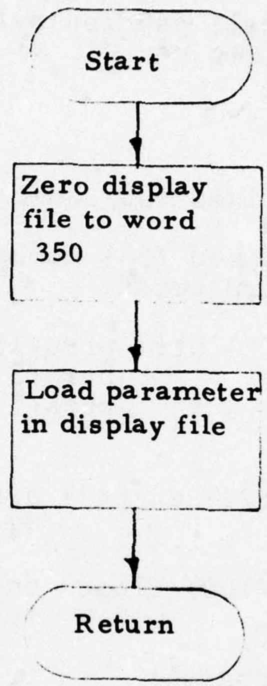
ndim	fixed (35) number of dimensions	
nn	fixed (35) number of nodes to be projected	
ptrs	(5) ptr	ptrs(1) - sysdata ptrs(3) - display file ptrs(5) - mooslogic file if it exists
low	(72) char(4)	array of node names to be projected
vl	(100) float "x" projection vector	
v2	(100) float "y" projection vector	
tc	char(1)	tree character of projected data set.
x	fixed (35) set to 1 if ss$display is called from logic design, 0 otherwise.	

Output File Settings: display is set up for a cluster or scatter plot.

Program Description: ss$display zeros, then initializes the display file by loading the given parameters according to standard two-space display file format.

Flow Chart: See following page

ss$display


Start
Zero display file to word 350
Load parameter in display file
Return

Internal Subroutine Name: ss$display1

Calling Sequence: call ss$display1 (nd, nn, ptrs, low, v1, tc, x)

Input Parameters:

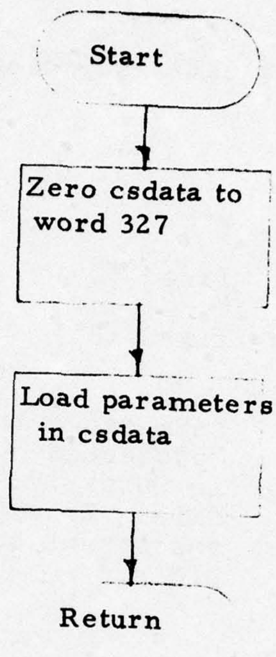
nd	fixed (35)	number of dimensions
nn	fixed (35)	number of nodes to be projected.
ptrs	(5) ptr	ptrs(1) - sysdata ptrs(3) - csdata ptrs(5) - mooslogic file if it exists
low	(72) char(4)	array of node names to be projected.
v1	(100) float	one-space projection vector.
tc	char(1)	tree character of projected data set.
x	fixed (35)	set to 1 if ss$display1 is called from logic design, 0 otherwise.

Output File Settings: csdata is set up for one-space display.

Program Description: ss$display1 zeros, then initializes the csdata file by loading the given parameters according to standard one-space display file format.

Flow Chart: See following page

ss$display1


Start
Zero csdata to word 327
Load parameters in csdata
Return

Internal Subroutine Name: ss$message

Calling Sequence: call ss$message (disptr, m1, m2)

Input Parameter:

disptr	ptr	pointer to display
m1	fixed (35)	measurement number
m2	fixed (35)	measurement number

Program Description: ss$message loads the phrase: "projected on meas (m1) and (m2)" in words 19 - 25 of the display file. This phrase appears under subsequent scatter or cluster plots.

Internal Subroutine Name: ss$message1

Calling Sequence: call ss$message1 (disptr, m1)

Input Parameters:

disptr	ptr	pointer to "csdata" file
m1	fixed (35)	measurement number

Program Description: ss$message 1 loads the phrase: "projected on meas. (m1)" in words 2 - 10 of the "csdata" file. This phrase appears under subsequent one-space plots.

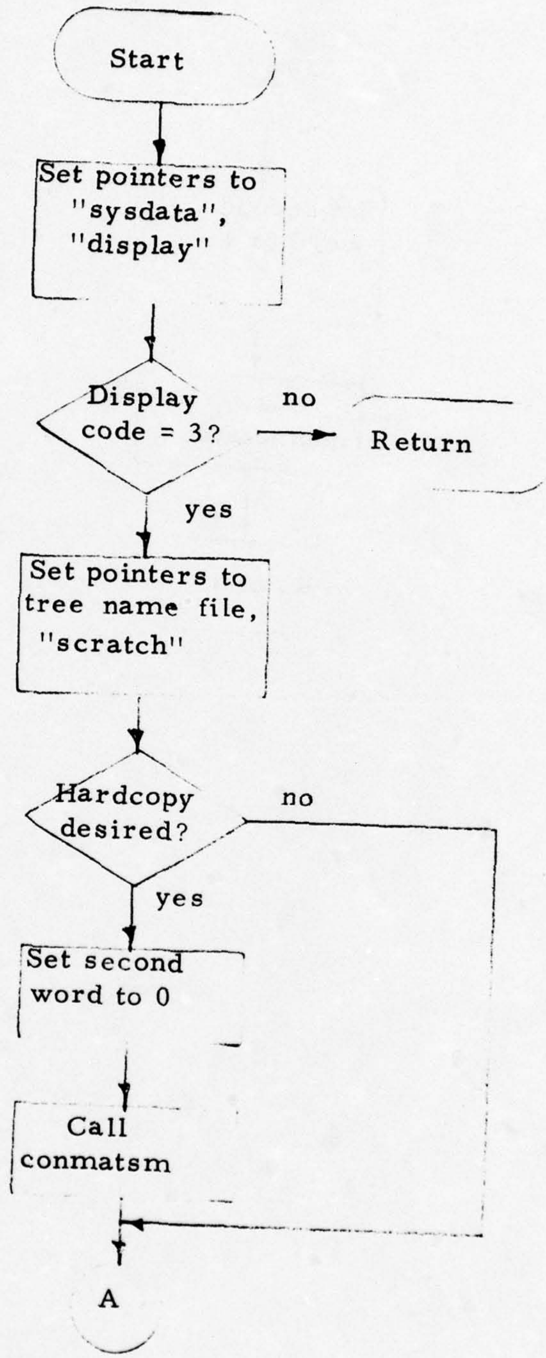
Utility Function Name: summrycm

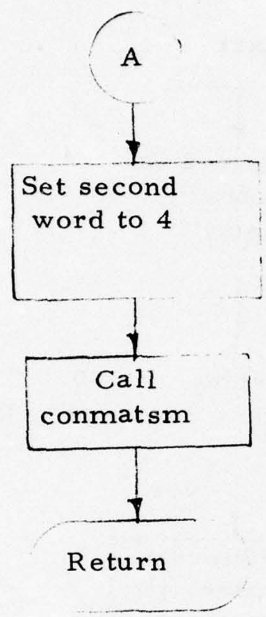
Calling Sequence: type in "summrycm"

Program Description: summrycm generates the four pointers if the display file code indicates a confusion matrix, determine if a hardcopy is desired and then calls conmatsm.

Flow Chart: See following page

summrycm


Start
Set pointers to "sysdata", "display"
Display code = 3?
no
Return
yes
Set pointers to tree name file, "scratch"
Hardcopy desired?
no
yes
Set second word to 0
Call conmatsm
A


A
Set second word to 4
Call conmatsm
Return

MOOS Function Name: tapeoput

MOOS Function Number: 101

Calling Sequence: type in "tapeoput [(treename)]"

Input Parameter:

treename	char(8)	tree to be written on tape

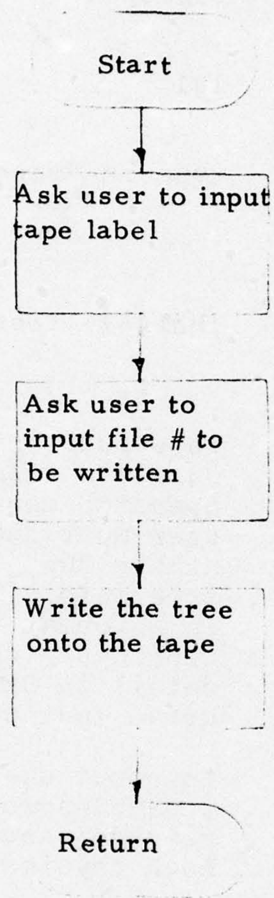
Program Description:

tapeoput first asks the user to supply a tape label for use in the operator mounting message. The user must then input the tape file number "he" wishes to write the tree into. The tree is written in a format compatible with tapeinput (described in more detail in Section 2). The tape drive must be 7 track, 556 B.P.I.

tapeoput uses subroutine t_save$herror to output error messages and t_read$record to read each physical tape record.

Flow Chart: See following page.

tapeoput


Start
Ask user to input tape label
Ask user to input file # to be written
Write the tree onto the tape
Return

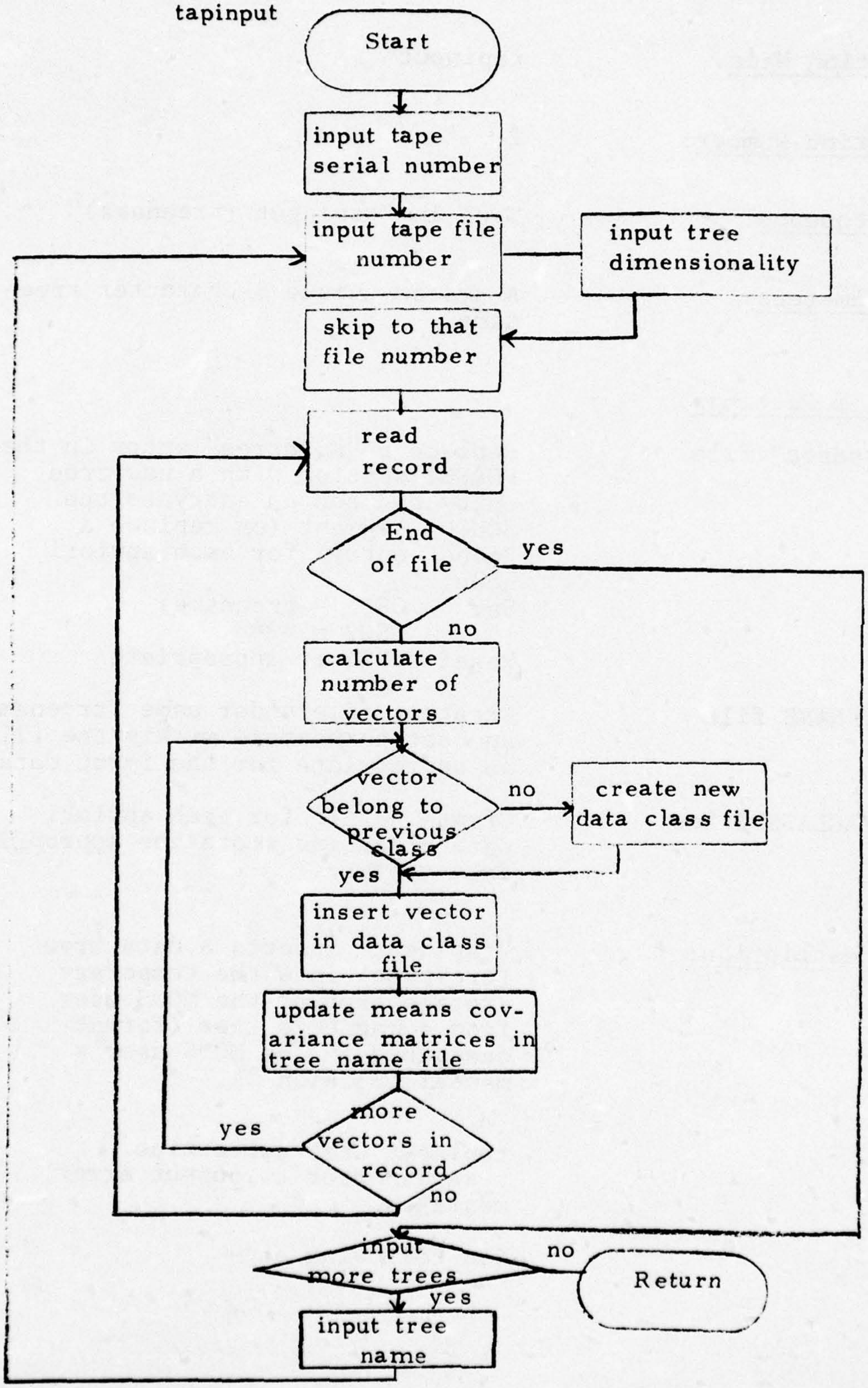
MOOS Function Name:	tapinput
MOOS Function Number:	2
Calling Sequence:	Type in "tapinput (treename)"
Input Parameters:	A system-unique 8-character tree-name.

Output File Settings:

"sysdata" file	Replace a "notatree" entry in the FOREST section with a new tree entry and add an entry to the SCHOOL segment (or replace a "nono" entry) for each apriori node. Set CSS1 = (treename) CSS2 = **** Reset CSS4 if appropriate
TREENAME file	Create a file under name "treename" and set parameters within the file as appropriate for the input data.
DATACLASS files	Create a file for each apriori data class and store the appropriate data vectors.

Program Description:	"tapinput" inserts a data tree (treename) into the temporary storage area of the MOOS user from a magnetic tape (format described in the MOOS user's manual, Section 2). tapinput uses subroutine t_save$herror to output error messages.
Flow Chart:	See following page

tapinput



Start
input tape serial number
input tape file number
input tree dimensionality
skip to that file number
read record
End of file
yes
no
calculate number of vectors
vector belong to previous class
no
create new data class file
yes
insert vector in data class file
update means covariance matrices in tree name file
more vectors in record
yes
no
input more trees
no
Return
yes
input tree name

Internal Subroutine Name: tfs

Calling Sequence: call tfs (tptr, nodename, index)

Input Parameters:

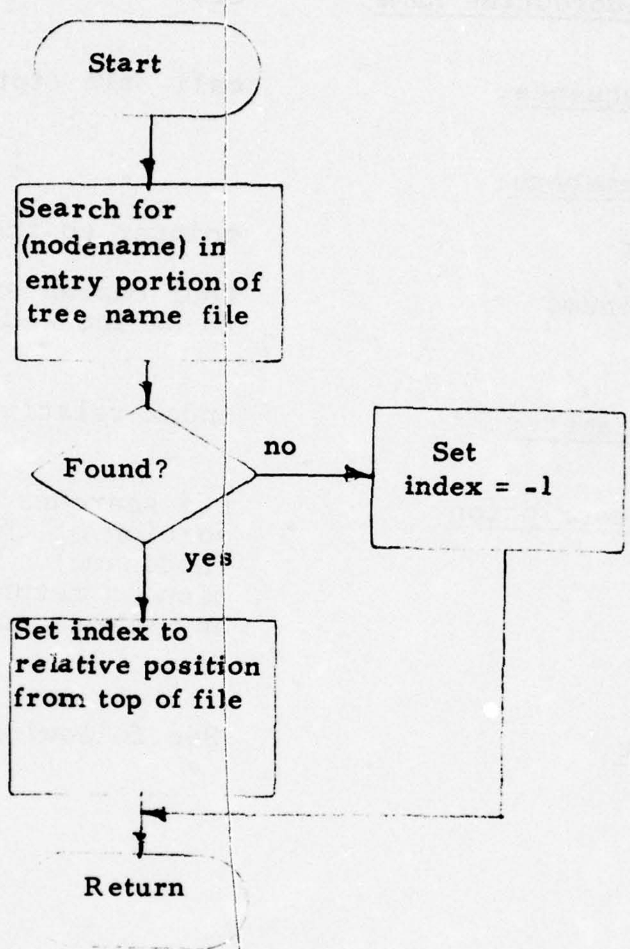
tptr	pointer to tree name file
nodename	four character node name of node to be located

Output Parameter: index-relative index of (nodename)

Program Description: tfs searches through the entry portion of the tree name file for (nodename). The relative position is returned in index. If any errors occur, index is set to -1.

Flow Chart: See following page

tfs


Start
Search for (nodename) in entry portion of tree name file
Found?
no
Set index = -1
yes
Set index to relative position from top of file
Return

Internal Subroutine Name: transgen

Calling Sequence: call transgen (exp, nexp)

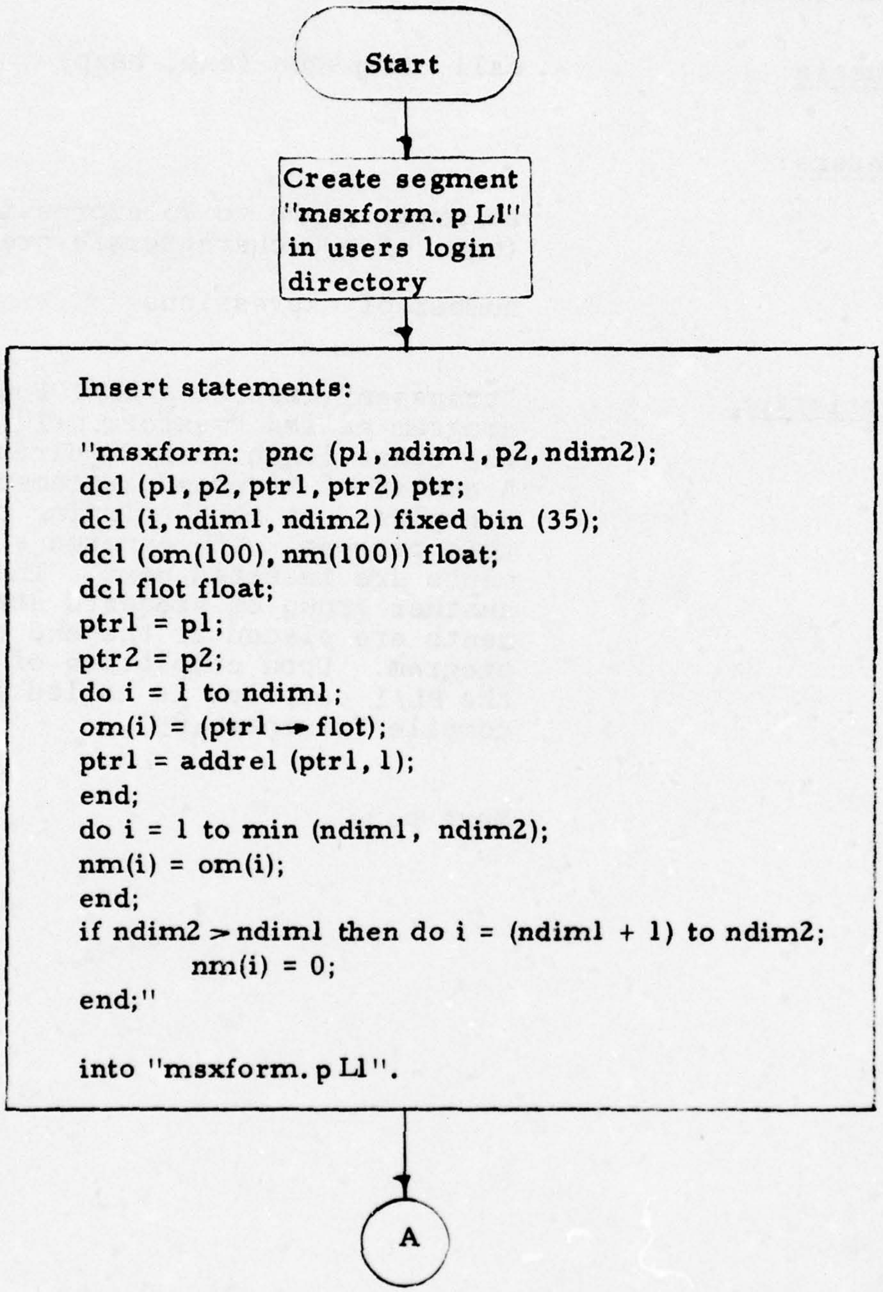
Input Parameters:

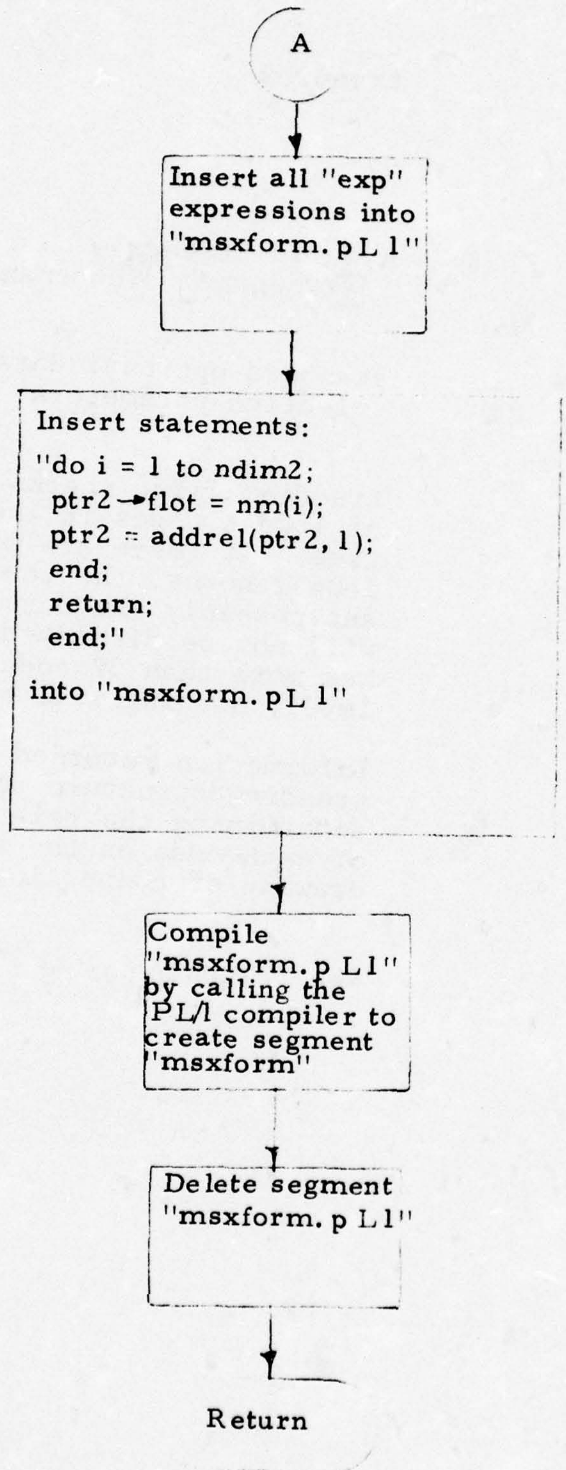
exp	an array of up to 75 expressions (max. of 300 characters/expression)
nexp	number of expressions

Program Description: "transgen" creates a PL/1 source program called "msxform.pℓ1" in the users login working directory. A number of standard statements are placed at the beginning of this program. The entered statements are inserted next. Then another group of standard statements are placed at the end of this program. Upon completion of this, the PL/1 compiler is called to compile "msxform.pℓ1".

Flow Chart: Next page

transgen


Start
Create segment "msxform. p Ll" in users login directory
Insert statements:
"msxform: pnc (pl, ndiml, p2, ndim2);
dcl (pl, p2, ptrl, ptr2) ptr;
dcl (i, ndiml, ndim2) fixed bin (35);
dcl (om(100), nm(100)) float;
dcl flot float;
ptrl = pl;
ptr2 = p2;
do i = 1 to ndiml;
om(i) = (ptrl → flot);
ptrl = addrel (ptrl, 1);
end;
do i = 1 to min (ndiml, ndim2);
nm(i) = om(i);
end;
if ndim2 > ndiml then do i = (ndiml + 1) to ndim2;
nm(i) = 0;
end;"
into "msxform. p Ll".
A


A
Insert all "exp" expressions into "msxform.pL1"
Insert statements:
"do i = 1 to ndim2;
ptr2→flot = nm(i);
ptr2 = addrel(ptr2, 1);
end;
return;
end;"
into "msxform.pL1"
Compile "msxform.pL1" by calling the PL/1 compiler to create segment "msxform"
Delete segment "msxform.pL1"
Return

MOOS Function Name: treedraw

MOOS Function Number: 22

Calling Sequence: Type in "treedraw [(treename)] [(nodename)] "

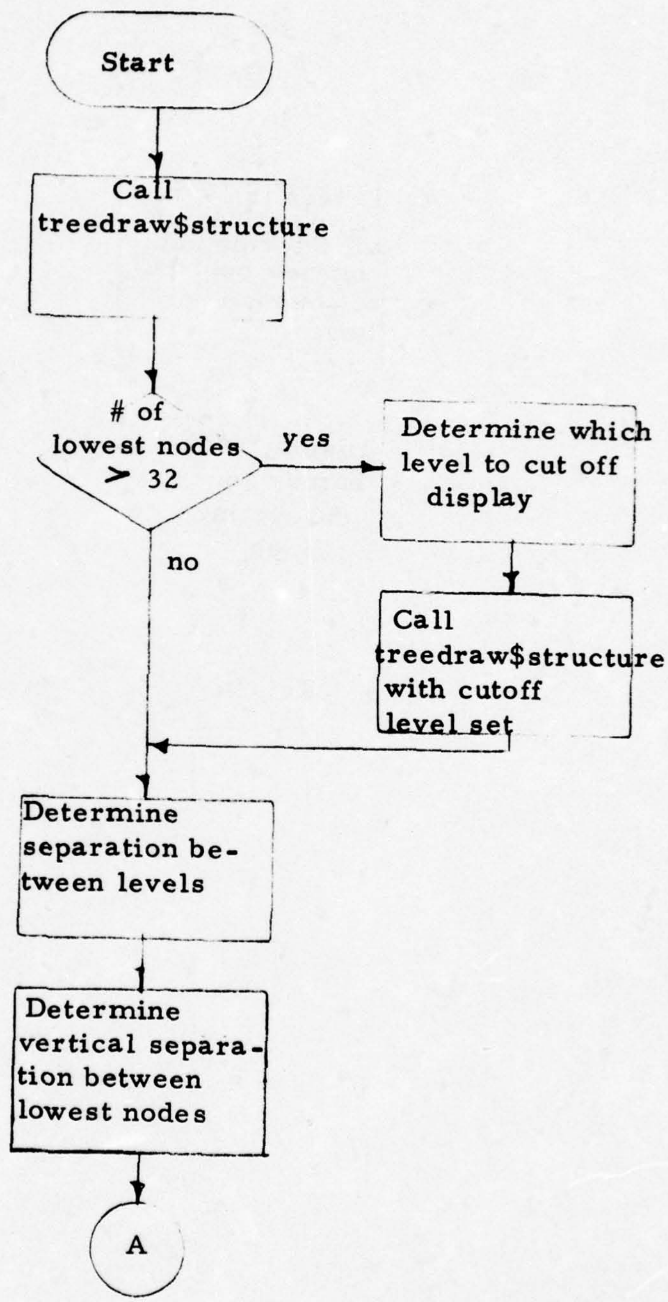
Input Parameters: standard optional data set selection parameters

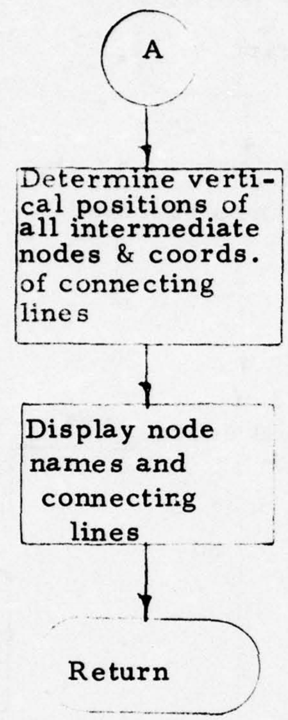
Program Description: treedraw first checks the number of lowest nodes in the selected tree. If there are more than 32 lowest nodes, the lowest level, and possibly the next lowest level will not be displayed. If a tree has more than 32 nodes on the first level, nothing will be drawn.

Information returned by treedraw$structure is used in determining the relative position of each node on the screen and the drawing of connecting lines.

Flow Chart: See following page

treedraw


Start
Call treedraw$structure
# of lowest nodes > 32
yes
Determine which level to cut off display
Call treedraw$structure with cutoff level set
no
Determine separation be- tween levels
Determine vertical separa- tion between lowest nodes
A


A
Determine vertical positions of all intermediate nodes & coords. of connecting lines
Display node names and connecting lines
Return

Internal Subroutine Name: treedraw$structure

Calling Sequence: call treedraw$structure (trnam, cnam, cln, fcl, cd, nl, nd, d, L, nn, nld)

Input Parameters:

trnam — char(8) tree name

cnam — char(4) node name

L — fixed (35) cutoff level. If set to 1 entire structure will be returned. Otherwise, L should be set to lowest level about which information is desired.

Output Parameters:

cln — (74) char(4) array of intermediate node names.

fcl — (74) fixed (35) array containing the "first class" (S4 in "sysdata") indices which correspond to the cln array.

cd — (74) fixed (35) array containing the depth of each intermediate node.

nl — (6) fixed (35) array containing number of nodes at each level.

nd — (72) char(4) array of lowest node names.

d — (72) fixed (35) array containing the depth of each lowest node.

L — fixed (35) lowest level found.

nn — fixed (35) number of intermediate nodes.

nld	fixed (35) number of lowest nodes.
Program Description:	treedraw$structure uses internal subroutines getclass and ut$getnode in a manner similar to lnodes to return the various arrays. If there are any errors, a message is printed and nn is set to -1. If a cut off level has been specified by setting L to a value other than 1, any nodes at the cut off level as well as lowest nodes above that level - are returned as lowest nodes.

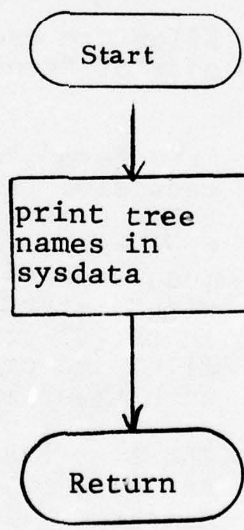
Utility Function Name: treelist

Calling Sequence: Type in "treelist"

Program Description: This routine lists the trees and nodes under these trees present in the "sysdata" file. This information is obtained from the forest and school sections of "sysdata".

Flow Chart:

treelist



MOOS Function Name: trnsform

MOOS Function Number: 7

Calling Sequence: Type in "trnsform"

Input File Settings: The routine assumes a rank-order display file format

Output File Settings:

"sysdata" — A new tree is created in sysdata

Treename File — A file under the name "treename" is created. Entries are made for the senior node and all other nodes of the tree "treename"

Data Class Files — Files are created for each lowest node of "treename"

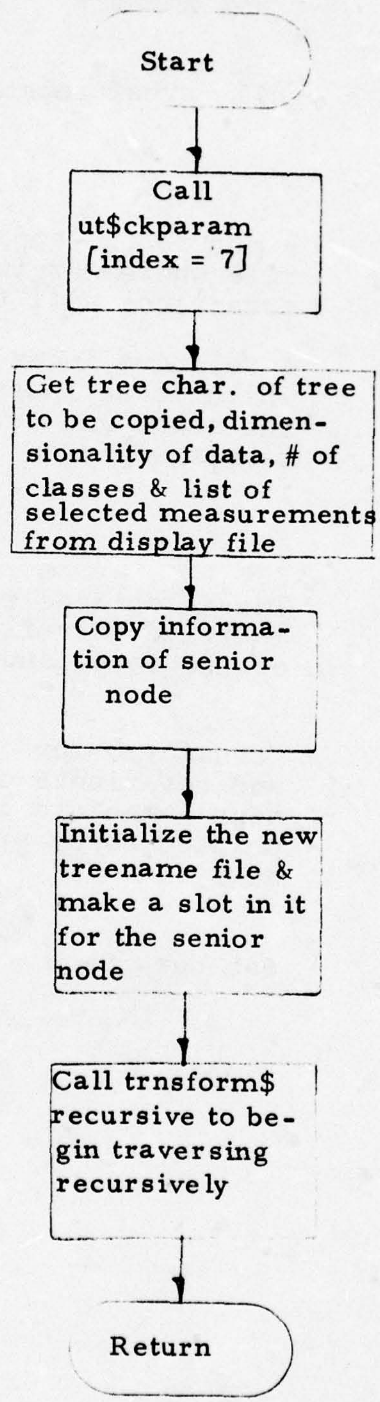
Program Description: "trnsform" along with "trnsform$recursive" and "trnsform$mmcv", takes a given tree and from this tree extracts a selected set of measurements (i.e. those measure- with a star associated with them in the d7 portion of the display file) and creates a new tree with these selected measurements.

The traversal of the given tree and copying of this routine is done recursively. The recursive rule used in traversing is:

```
visit the root
do i = 1 to number of subtrees of
        root, traverse the ith sub-
        tree of root
end
```

Where visiting the root means (in this case) copying the node and if a lowest create a data class file.

Flow Chart: trnsform



Internal Subroutine Name: trnsform$mmcv

Calling Sequence: call trnsform$mmcv (ptr, sysix)

Input Parameters:

ptr — A ptr to the top of the tree name file where merging of means and covariance will take place

sysix — A relative index into sysdata which is an index to a node where the means and covariance are to be merged into

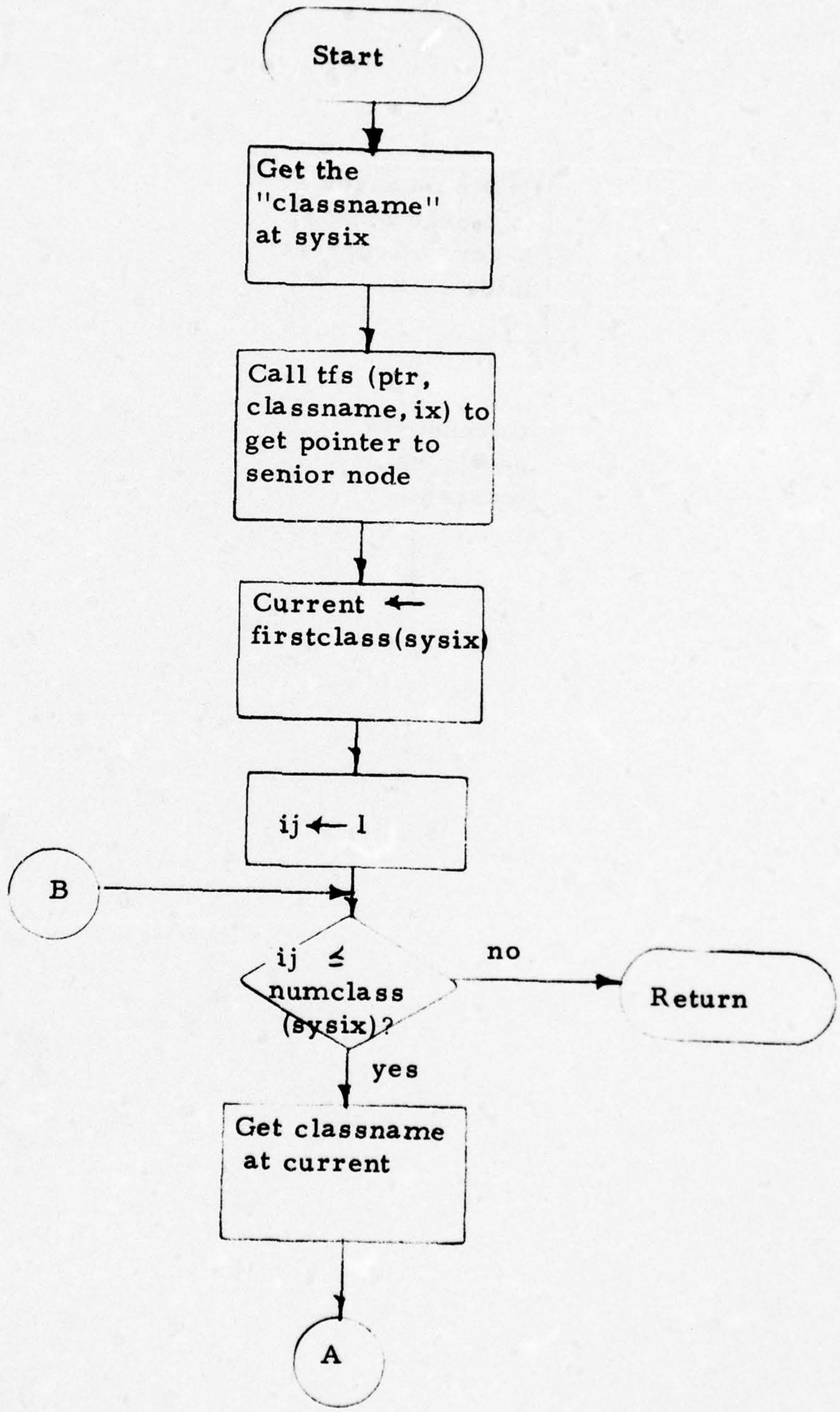
Output File Settings:

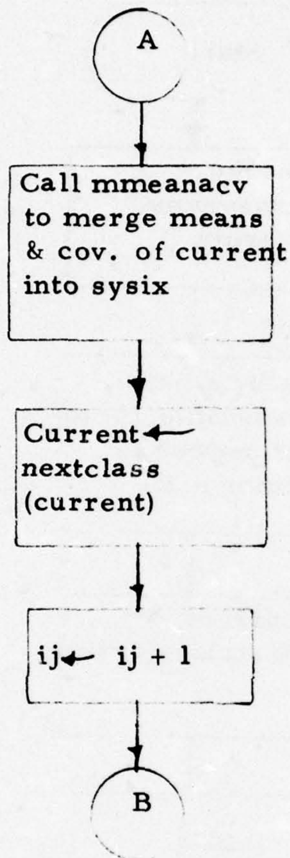
Treename File — The associated tree name file is modified to reflect the merging of the means and covariance

Program Description: "trnsform$mmcv" merges the means and covariance of all nodes at the next immediate level below the node pointed to by sysix into the node at sysix

Flow Chart: See next page

trnsform$mmcv




A
Call mmeanacv to merge means & cov. of current into sysix
Current ← nextclass (current)
ij ← ij + 1
B

Internal Subroutine Name: trnsform$recursive

Calling Sequence: call trnsform$recursive (current, newcurrent, numdim, lencov, list, ntptr, otptr, tptr, sptr, otreechar, ntreechar, ji, ndim)

Input Parameters:

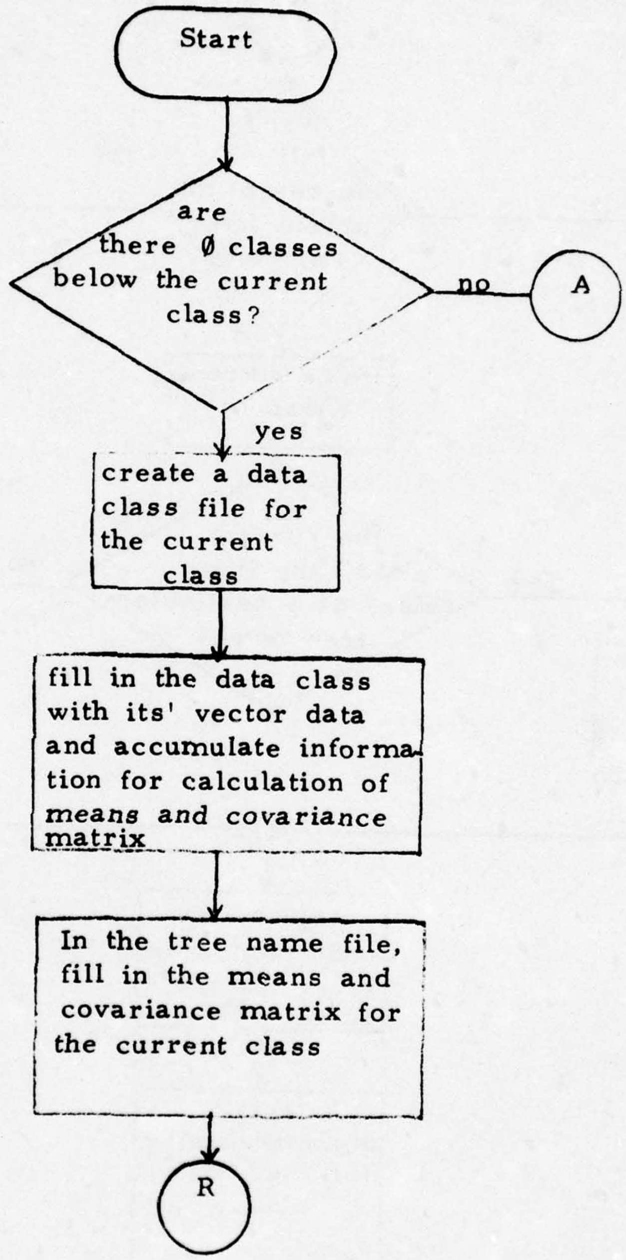
current	an index into sysdata of the current node of the tree being copied
newcurrent	an index into sysdata of the copy of current
numdim	the dimensionality of the data in the copied tree
lencov	the length of the covariance matrix in the copied tree
list	an array of size numdim which contains the measurement numbers that are to be extracted from the tree being copied and placed into the copied tree
ntptr	a ptr to the top of the new tree-name file
otptr	a ptr to the top of the old tree-name file
tptr	a current ptr into the new tree-name file and it is a ptr to the next available slot
sptr	a pointer to sysdata
otreechar	the tree character of the tree being copied
ntreechar	the tree character of the copied tree
ji	=3 if senior node of tree =2 otherwise
ndim	dimensionality of data in tree to be copied

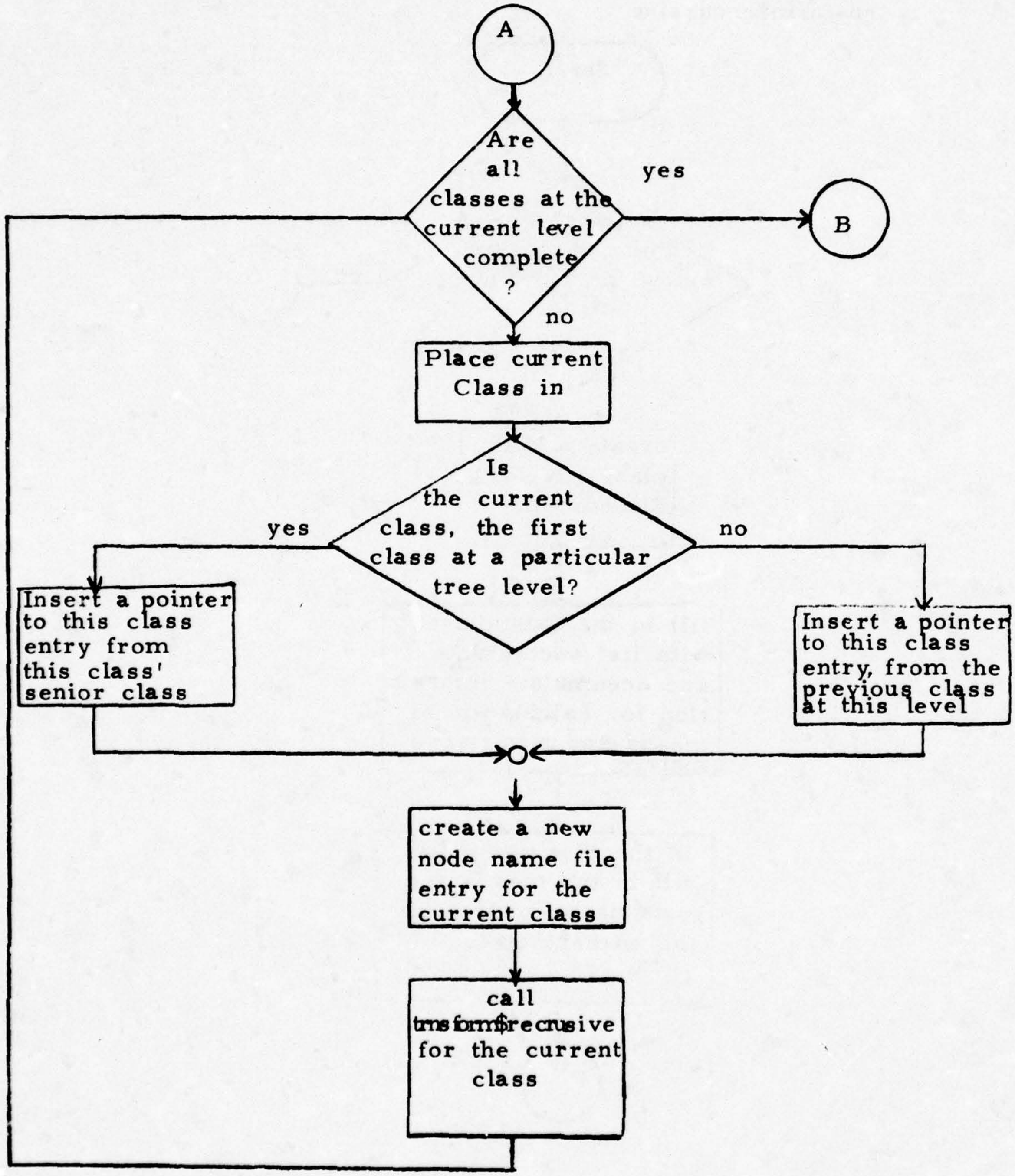
Output File Settings: same as "trnsform"

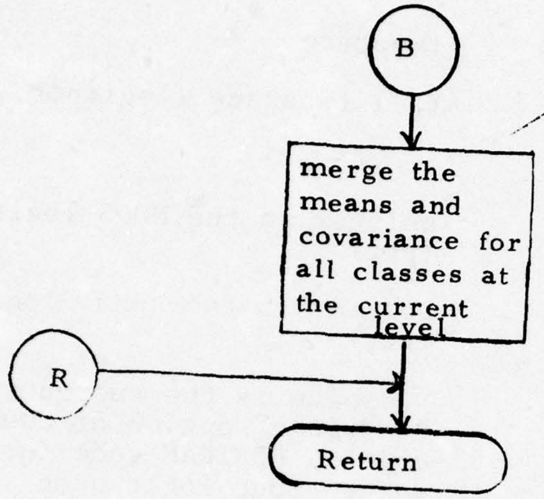
Program Description: "trnsform$recursive" is the part of trnsform which does the recursion of copying the tree

Flow Chart: See following page

trnsform$recursive




A
Are all classes at the current level complete ?
yes
B
no
Place current Class in
Is the current class, the first class at a particular tree level?
yes
no
Insert a pointer to this class entry from this class' senior class
Insert a pointer to this class entry, from the previous class at this level
create a new node name file entry for the current class
call
trnsformrecrusive
for the current class


B
merge the means and covariance for all classes at the current level
R
Return

Internal Subroutine Name: twospace

Calling Sequence: call twospace (logicptr, nodeptr)

Input Parameters:

logicptr — pointer to the MOOS logic file (ptr)

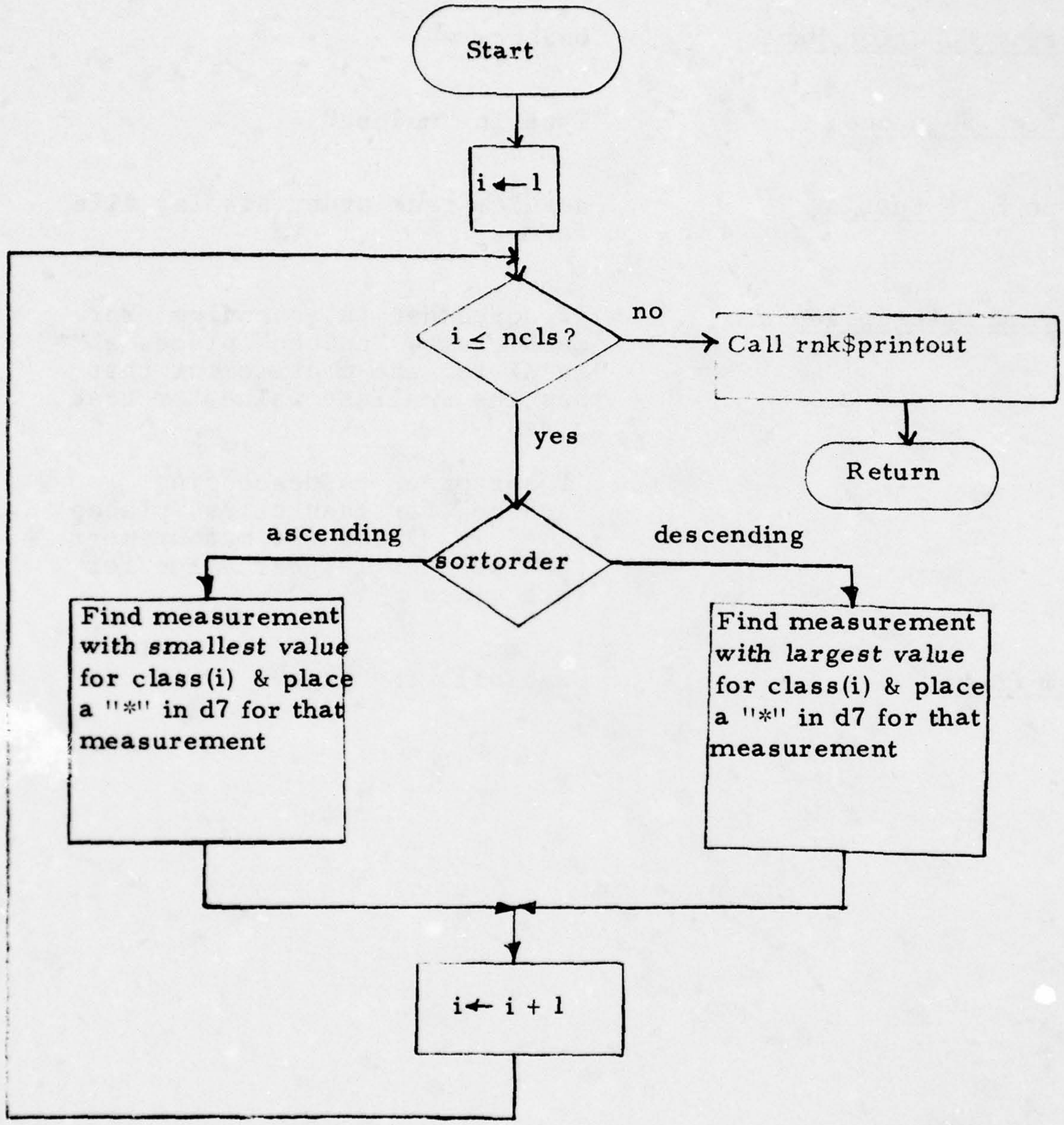
nodeptr — pointer to two-space logic node (ptr)

Program Description: twospace is the subroutine in the "fortlogc" option of MOOS which creates FORTRAN code for a two-space group logic node

See the subroutine's program listing for a more detailed description of the operation of this subroutine.

Utility Function Name:	un$bbc
Calling Sequence:	Type in "un$bbc"
Input File Setting:	assumes rank order display file format
Program Description:	if sortorder is ascending, for each class, "un$bbc" places a "*" in d7 for the measurement that has the smallest value for that class if sortorder is descending, "un$bbc" for each class, places a "*" in d7 for the measurement that has the largest value for that class
Flow Chart:	See following page

un$bbc



Start
i ← 1
i ≤ ncls?
no
Call rnk$printout
Return
yes
ascending
sortorder
descending
Find measurement with smallest value for class(i) & place a "*" in d7 for that measurement
Find measurement with largest value for class(i) & place a "*" in d7 for that measurement
i ← i + 1

Utility Function Name: un$bbcp

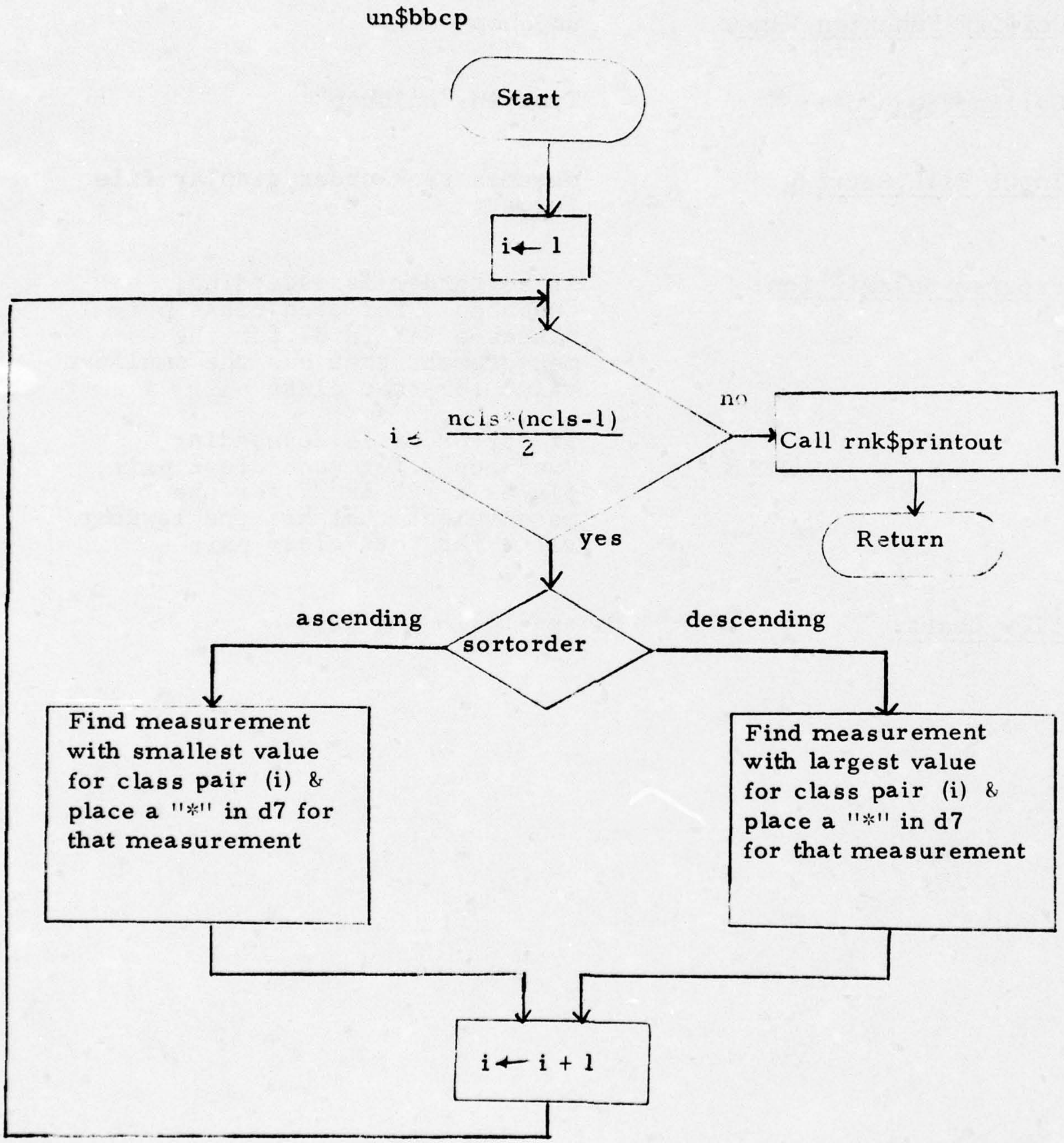
Calling Sequence: Type in "un$bbcp"

Input File Setting: assumes rank order display file format

Program Description: if sortorder is ascending, "un$bbcp", for each class pair places a "*" in d7 for the measurement that has the smallest value for that class pair

if sortorder is descending, "un$bbcp", for each class pair, places a "*" in d7 for the measurement that has the largest value for that class pair

Flow Chart: see following page


un$bbcp
Start
i ← 1
i ≤ ncls*(ncls-1) / 2
no
Call rnk$printout
Return
yes
sortorder
ascending
descending
Find measurement with smallest value for class pair (i) & place a "*" in d7 for that measurement
Find measurement with largest value for class pair (i) & place a "*" in d7 for that measurement
i ← i + 1

Internal Subroutine Name: ut$ckparam

Calling Sequence: call ut$ckparam (index, ptrs, treename, nodename)

Input Parameter Settings: Before entry into this routine two variables must be set. The variable index must contain the MOOS option number to be placed in the sysdata file (CSS5) and ptrs must point to the argument list that was entered at the command level. This list will be used to insert new names into the current treename (CSS1) and current classname (CSS2) in the sysdata file. The pointer may be set by using the subroutine cu_$arg_list_ptr (ptrs(1)) prior to entry into ckparam.

Output File Settings: The sysdata file will be updated with the current option number, the current tree name and the current class name and the current tree name file will contain the names of the lowest nodes (L5).

Program Description: Upon entry the sysdata file is initiated. If it does not exist, moosinitiate is called and sysdata is again initiated. The number of arguments is counted and the index is checked for a value less than 20. If this case exists, a tree name is expected in the command parameter list and this tree name is inserted into the FOREST, the class name is set to the senior class (****), and a new tree name segment is created.

Then, if the number of optional arguments equals one, the parameter will be retrieved and determined to be either a tree name (5-8 characters), or a class name (4 characters). This name will be inserted into CSS1 or CSS2 of the sysdata file. Then the lowest node will be found using this node as the senior node. If the argument is a class name, the class must exist somewhere in the current tree SCHOOL.

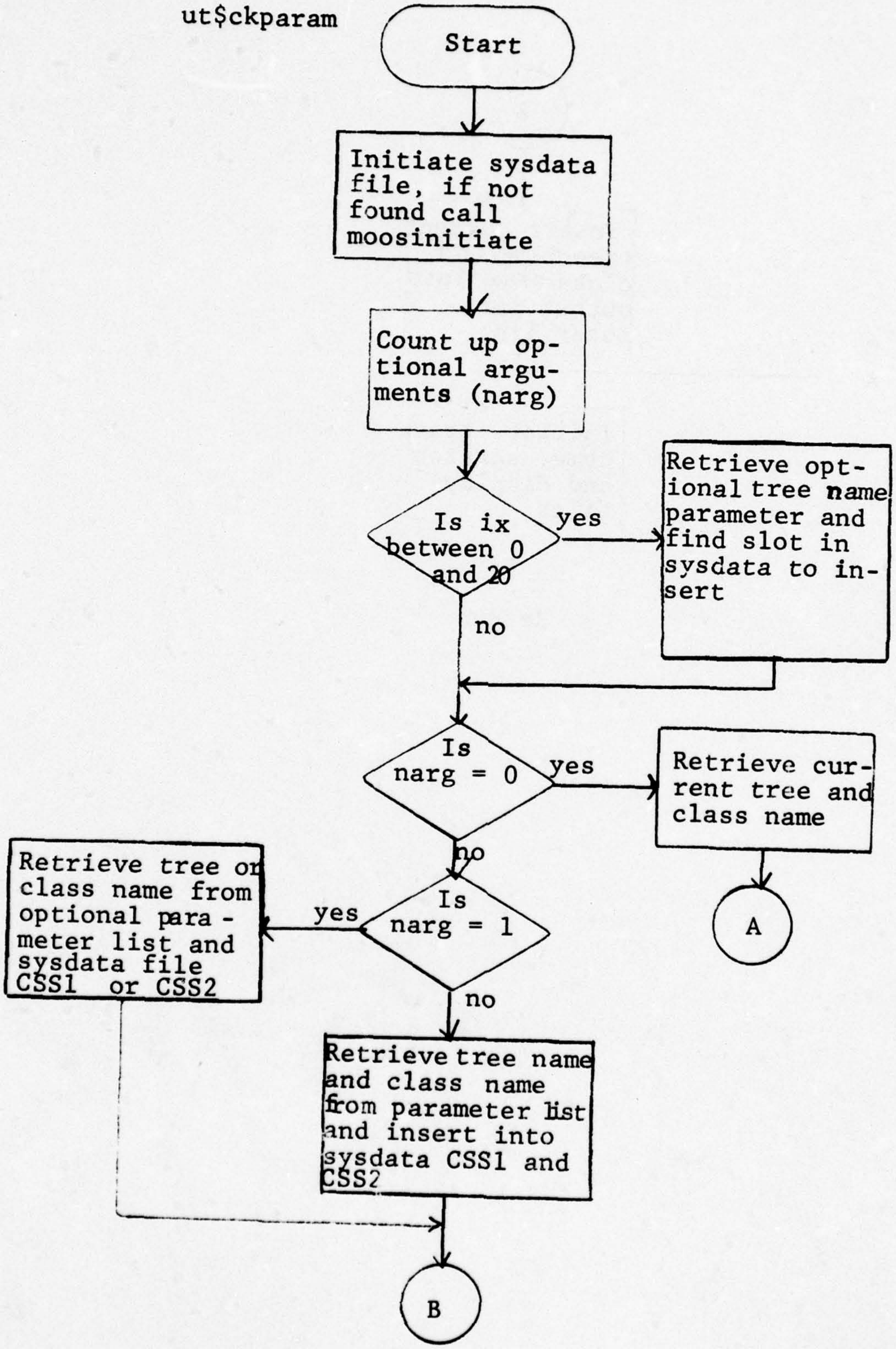
If two optional arguments are input, both a tree name and class name are retrieved, inserted in CSS1 and CSS2 of sysdata, and the lowest nodes are found using the class name as the senior node. Again the class name must exist somewhere in the SCHOOL and be associated with the current tree.

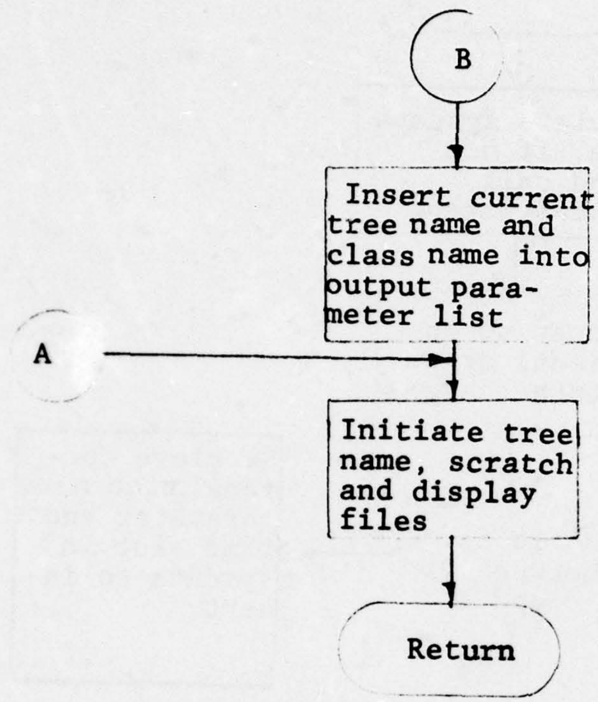
If there are no optional arguments, the current tree name and class name will be retrieved. Before the routine exits, tree name is set to the current class and the four pointers are set as follows:

ptrs(1) points to the "sysdata" file
ptrs(2) points to the "scratch" file
ptrs(3) points to the "display" file
ptrs(4) points to the current tree name file

<u>Flow Chart</u>: See following page

ut$ckparam


Start
Initiate sysdata file, if not found call moosinitiate
Count up optional arguments (narg)
Is ix between 0 and 20
yes
Retrieve optional tree name parameter and find slot in sysdata to insert
no
Is narg = 0
yes
Retrieve current tree and class name
A
no
Is narg = 1
yes
Retrieve tree or class name from optional parameter list and sysdata file CSS1 or CSS2
no
Retrieve tree name and class name from parameter list and insert into sysdata CSS1 and CSS2
B


B
Insert current tree name and class name into output parameter list
A
Initiate tree name, scratch and display files
Return

Internal Subroutine Name: ut$getnode

Calling Sequence: call ut$getnode(index,name,array)

Input Parameter Settings: index must be set to the relative word in sysdata of the node to be retrieved.

name must be set to the name in the SCHOOL of the node to be retrieved if and only if index is set to zero, otherwise it is a dummy variable.

Program Description: The routine first initializes the "sysdata" file and then tests index. If zero, the routine searches the SCHOOL portion of the file until it finds the node specified by the variable name in the parameter list. The relative word address will be placed in index and the node will be retrieved.

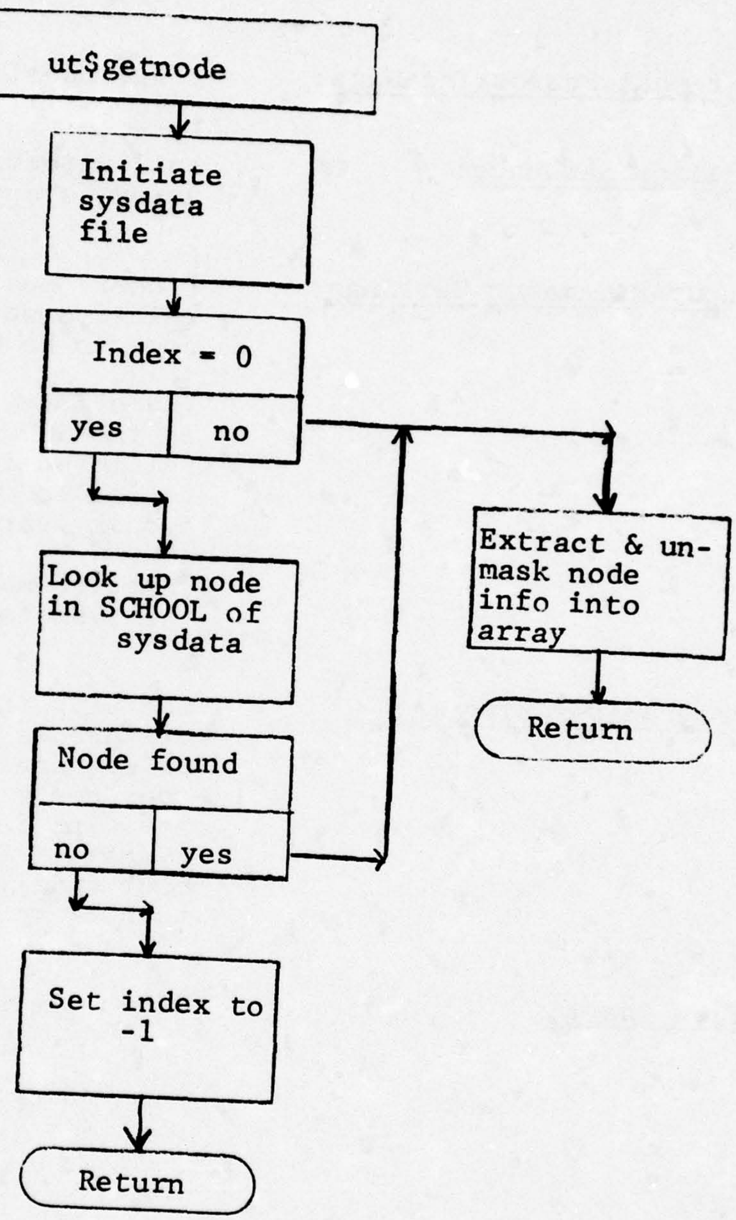
If the value of index is not zero, it will use this value as the relative word address into the sysdata file for the node to retrieve.

If the index is between 64 and 140, the contents of node in the FOREST will be returned in the variable "array", and if the index is greater than or equal to 144 the contents of a node in SCHOOL will be returned.

In the case of a FOREST node the first and second word of the array will be the name of the node, the next four words will be the number of dimensions, the number of classes below this node, the index of the first class below this node and the number of vectors in this node. The last two words are set to zero.

In the case of a SCHOOL node, the first word will be the class name and the next seven words will be the number of dimensions, the number of classes below this node, the index of the first class below this node, the number of vectors in this node, the depth of this node within the tree structure, the index of the senior class node for this node, and the index for the next class node in this level.

If any errors occur, the index is set to negative one and the routine exits.


ut$getnode
Initiate sysdata file
Index = 0
yes
no
Look up node in SCHOOL of sysdata
Node found
no
yes
Set index to -1
Return
Extract & un-mask node info into array
Return

Internal Subroutine Name: ut$getnodel

Calling Sequence: call ut$getnodel (index, name, array, sysptr)

Input Parameter Settings: "index" must be set to the relative word in sysdata of the node to be retrieved.

"name" must be set to the name in the SCHOOL of the node to be retrieved if and only if index is set to zero, otherwise it is a dummy variable.

"sysptr" must be a pointer to the sysdata file.

Program Description: See ut$getnode. The major difference between ut$getnode and ut$getnodel that is ut$getnodel does not make a call to hcs_$initiate to initiate sysdata. The calling program must initiate sysdata and pass a pointer to sysdata to ut$getnodel.

Flow Chart: See ut$getnode

Internal Subroutine Name: ut$putnode

Calling Sequence: call ut$putnode (index, name, array)

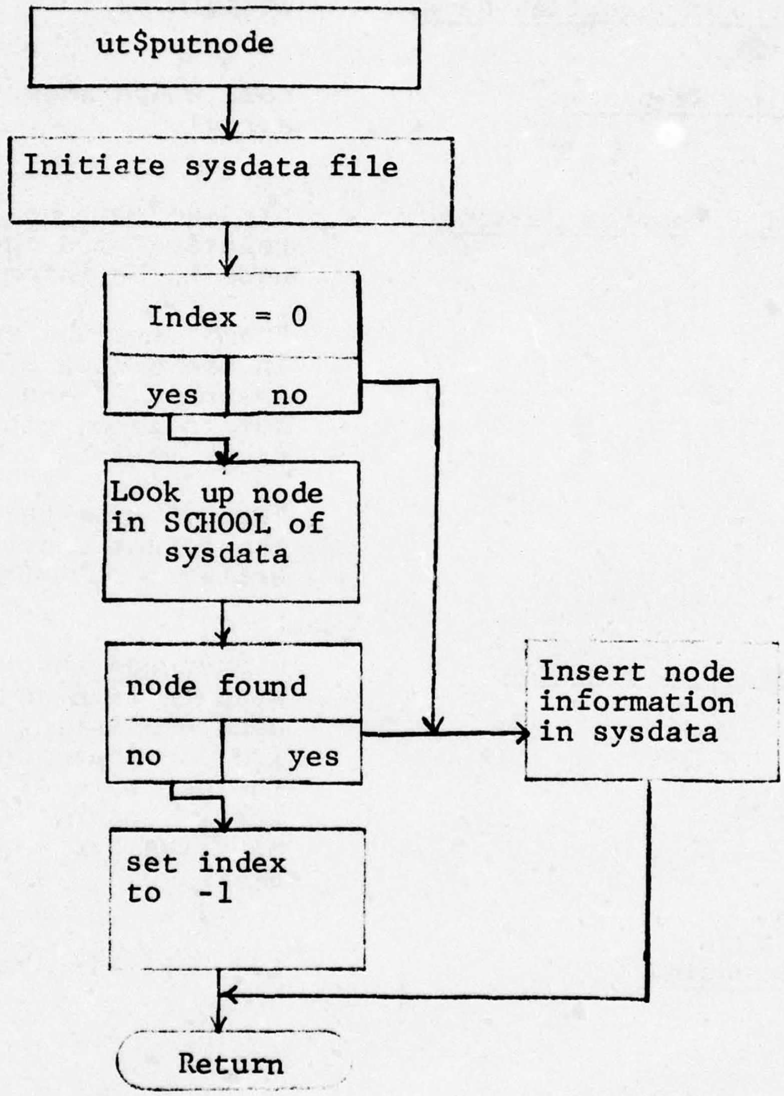
Input Parameter Settings: "index" must be set to the relative word in sysdata of the node to be inserted.

"name" must be set to the name in the SCHOOL of the node to be inserted if and only if index is set to zero, otherwise it is a dummy variable.

"array" must be set according to the format described in the write up on ut$getnode.

Program Description: ut$putnode inserts a node in the sysdata file and may be used to update existing node information, i.e., ut$putnode is essentially the opposite of ut$getnode. If any errors occur, "index" is set to negative one and the routine exits.

Flow Chart: See following page.


ut$putnode
Initiate sysdata file
Index = 0
yes
no
Look up node in SCHOOL of sysdata
node found
no
yes
Insert node information in sysdata
set index to -1
Return

Internal Subroutine Name: ut$putnodel

Calling Sequence: call ut$putnodel (index, name, array, sysptr)

Input Parameter Settings: "index" must be set to the relative word in sysdata of the node to be inserted.

"name" must be set to the name in the SCHOOL of the node to be inserted if and only if index is set to zero, otherwise it is a dummy variable.

"array" must be set according to the format described in the write-up on ut$getnode.

"sysptr" must be a pointer to the sysdata file.

Program Description: See ut$putnode. The major difference between ut$putnode and ut$putnodel is that ut$putnodel does not make a call to hcs_$initiate to initiate sysdata. The calling program must initiate sysdata and pass a pointer to sysdata to ut$putnodel.

Flow Chart: See ut$putnode

Utility Subroutine Name: vec$del

Calling Sequence: type in "vec$del (vname)"

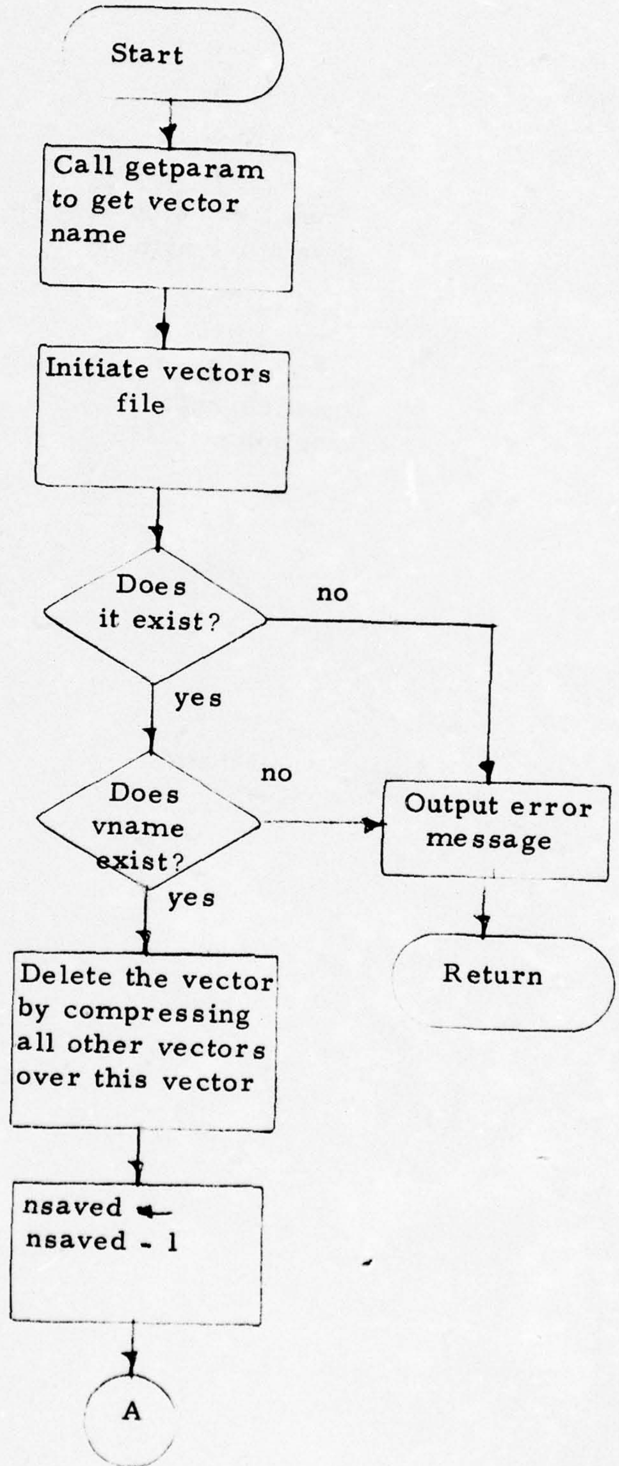
Input Parameter:

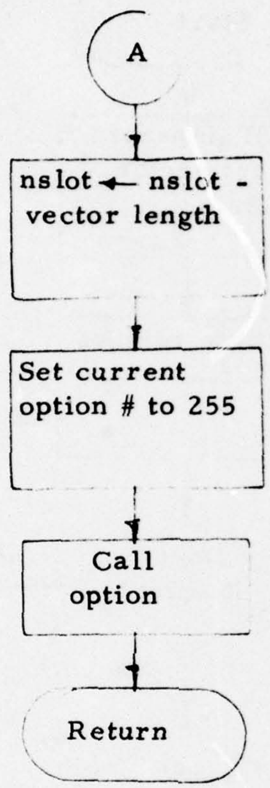
vname — the name of the vector to be deleted

Program Description: "vec$del" deletes a vector from the users "vectors" file

Flow Chart: see following page

vec$del


Start
Call getparam to get vector name
Initiate vectors file
Does it exist?
no
yes
Does vname exist?
no
yes
Output error message
Return
Delete the vector by compressing all other vectors over this vector
nsaved ←
nsaved - 1
A


A
nslot ← nslot - vector length
Set current option # to 255
Call option
Return

Internal Subroutine Name: vec$get

Calling Sequence: call vec$get (vname, vlength, vptr)

Input Parameters:

vname — the name of the vector to be retrieved char(8)

vlength — the length of this vector fixed bin (35)

vptr — a ptr to where the retrieved vector is to be stored

Output Parameter:

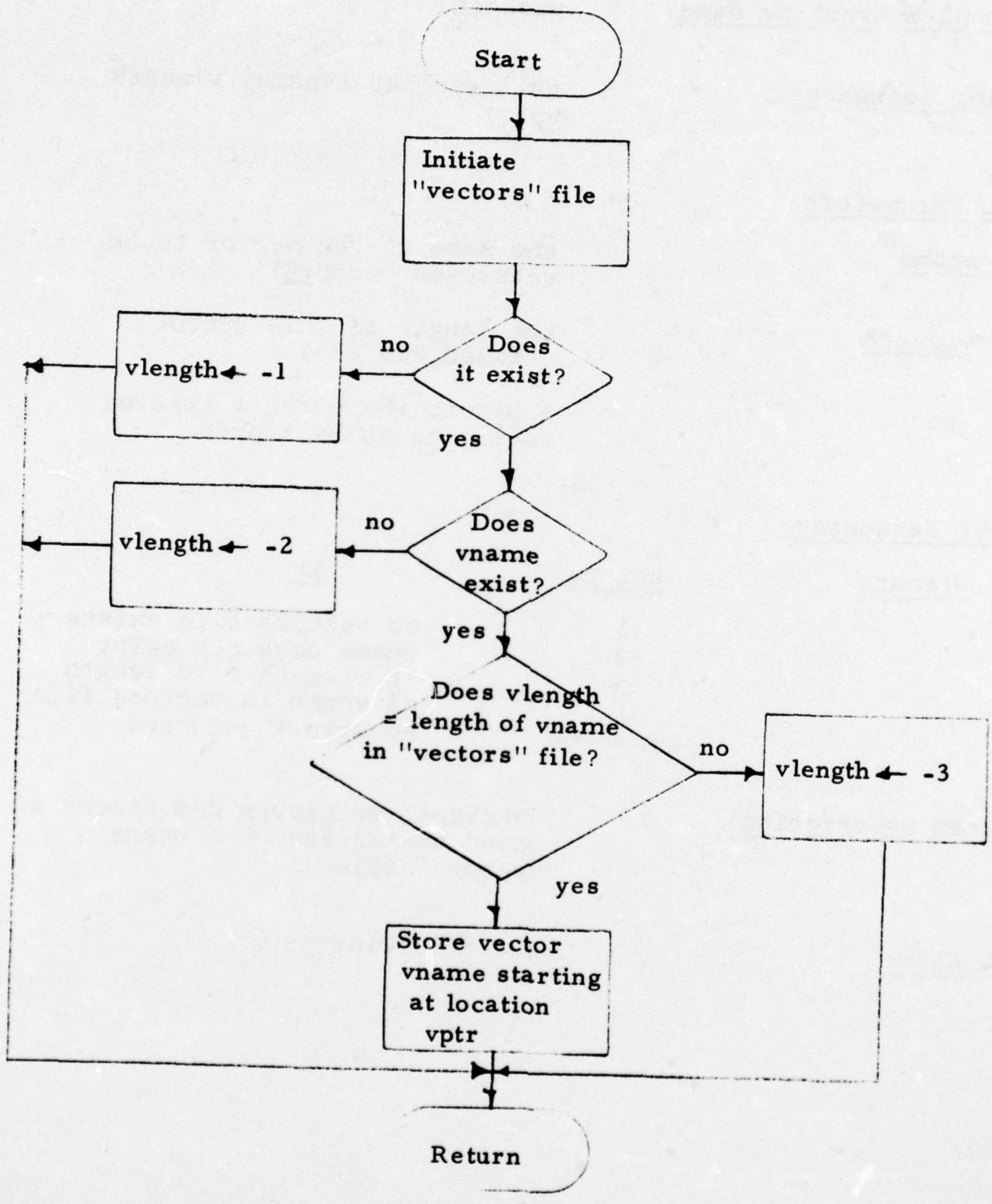
vlength

set to	if
-1	no vectors file exists
-2	vname does not exist
-3	if vlength ≠ to length of vname in vectors file
unchanged	no errors occurred

Program Description: "vec$get" retrieves and stores a saved vector from the users "vectors" file

Flow Chart: see following page

vec$get


Start
Initiate "vectors" file
Does it exist?
no
vlength ← -1
yes
Does vname exist?
no
vlength ← -2
yes
Does vlength = length of vname in "vectors" file?
no
vlength ← -3
yes
Store vector vname starting at location vptr
Return

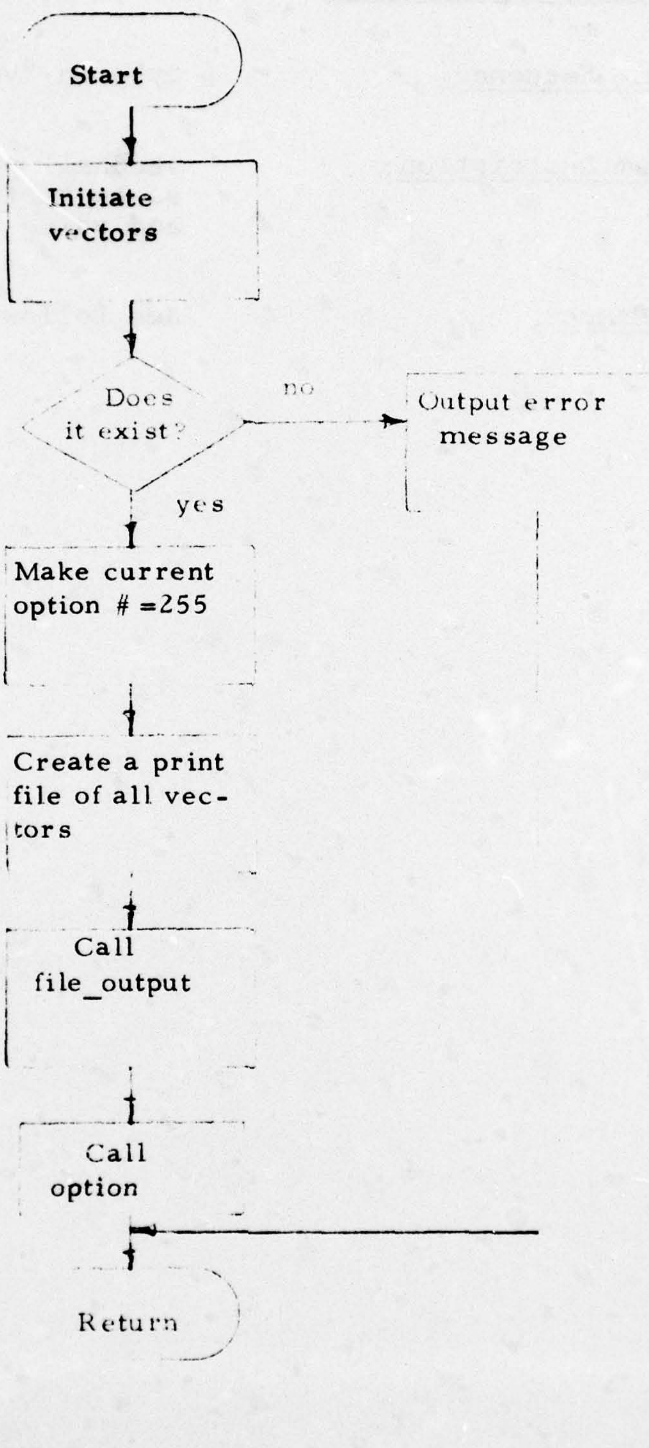
Utility Subroutine Name: vec$hall

Calling Sequence: type in "vec$hall"

Program Description: vec$hall prints or hardcopies all saved vectors, their name, length, and value

Flow Chart: see following page

vec$hall



Start
Initiate vectors
Does it exist?
no
Output error message
yes
Make current option # =255
Create a print file of all vectors
Call file_output
Call option
Return

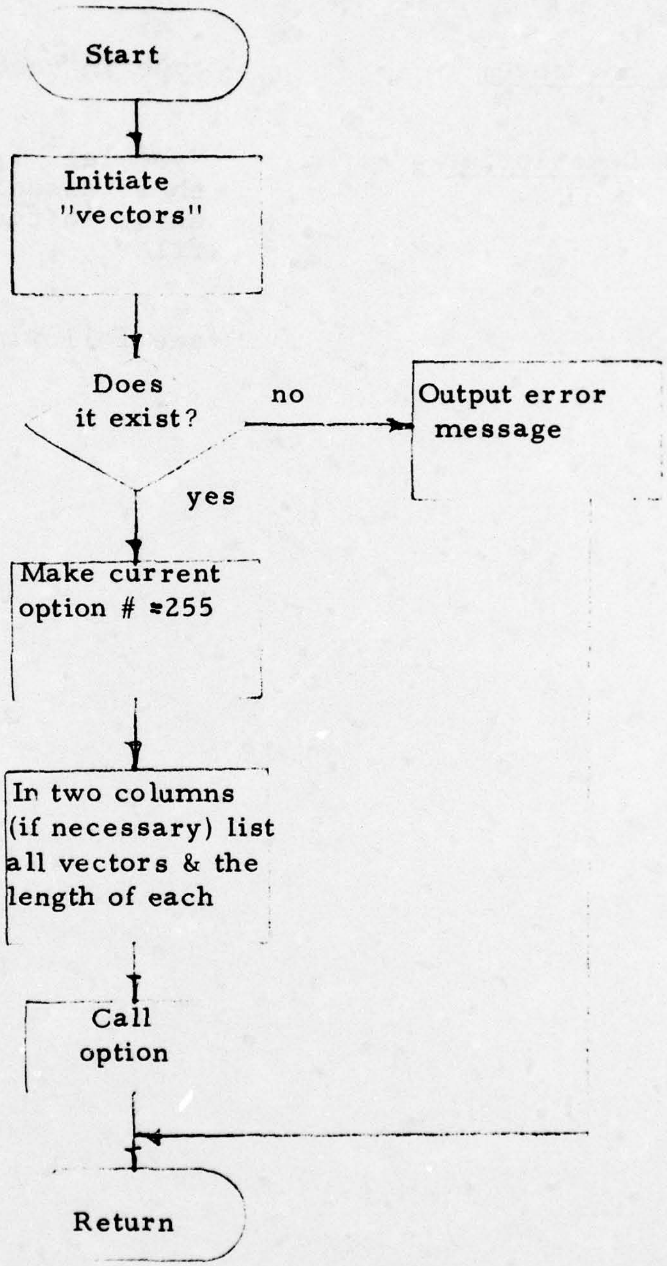
Utility Subroutine Name: vec$lall

Calling Sequence: type in"vec$lall"

Program Description: "vec$lall" list all vectors and their associated length that exist in the users home vectors file

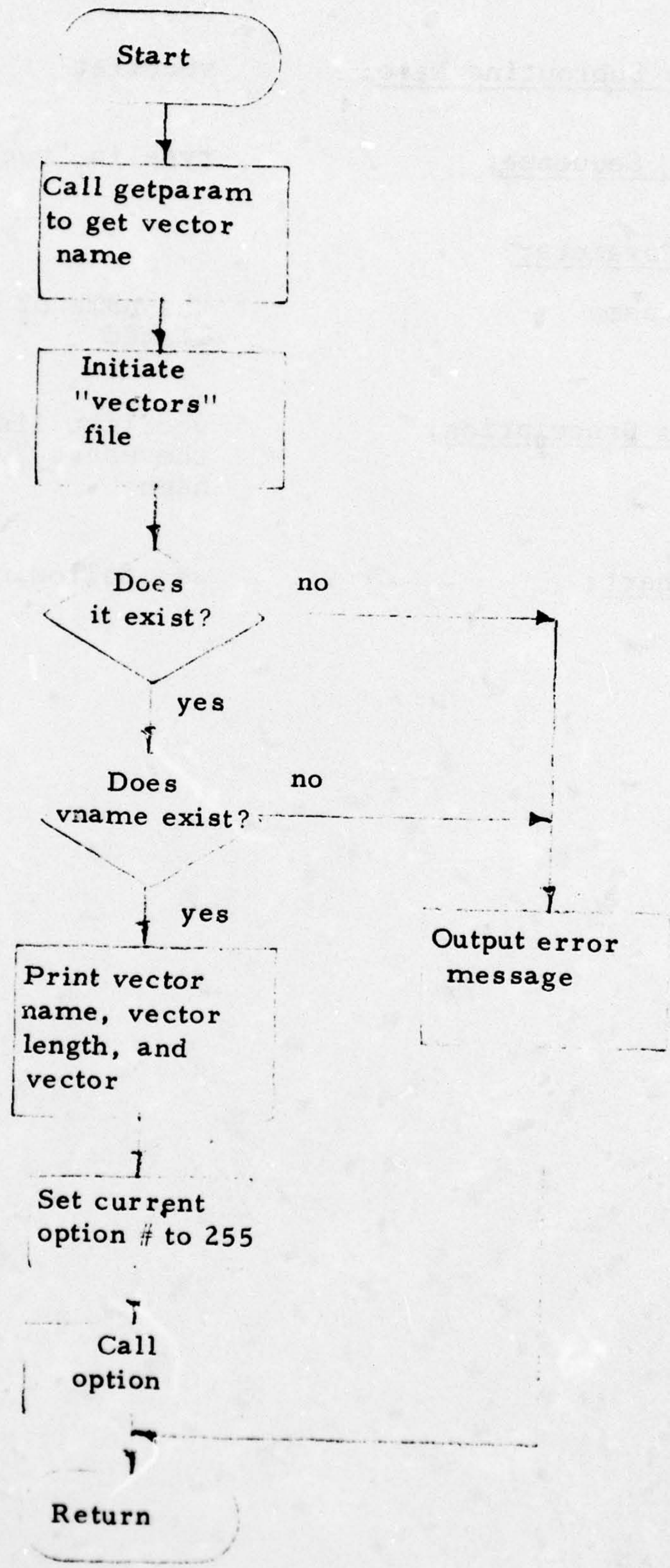
Flow Chart: see following page

vec$lall


Start
Initiate
"vectors"
Does
it exist?
no
Output error
message
yes
Make current
option # =255
In two columns
(if necessary) list
all vectors & the
length of each
Call
option
Return

Utility Subroutine Name:	vec$list
Calling Sequence:	type in "vec$list (vname)"
Input Parameter:	
vname	the name of the vector to be listed
Program Description:	vec$list lists a vector from the users "vectors" file by name
Flow Chart:	see following page

vec$list


Start
Call getparam to get vector name
Initiate "vectors" file
Does it exist?
no
yes
Does vname exist?
no
yes
Print vector name, vector length, and vector
Output error message
Set current option # to 255
Call option
Return

Utility Function Name: vec$save

Calling Sequence: type in "vec$save"

Input File Setting:

display	a scatter-cluster or a histogram display file format must exist

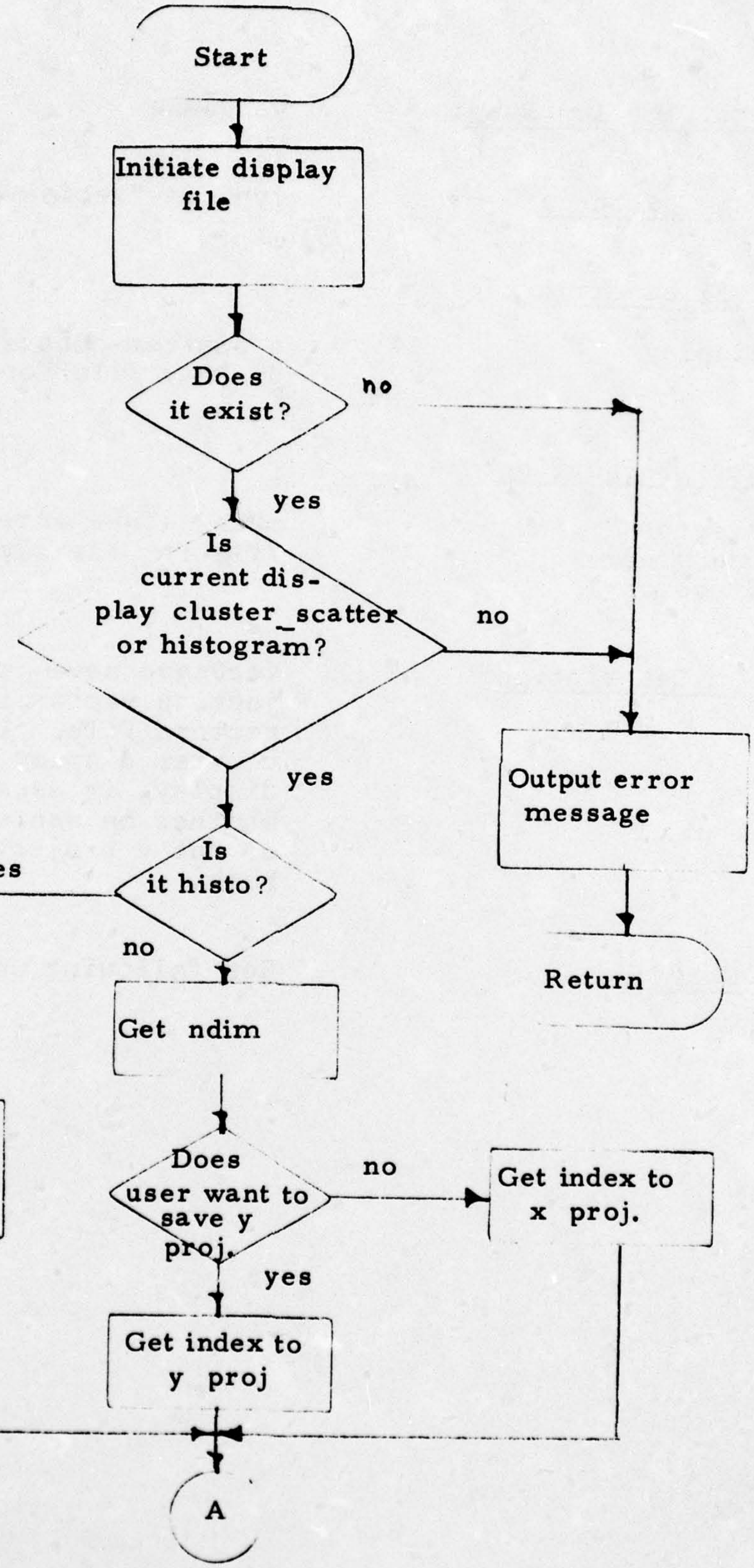
Output File Setting:

"vectors" (in users home directory)	makes a new entry in the vectors file for the saved vector

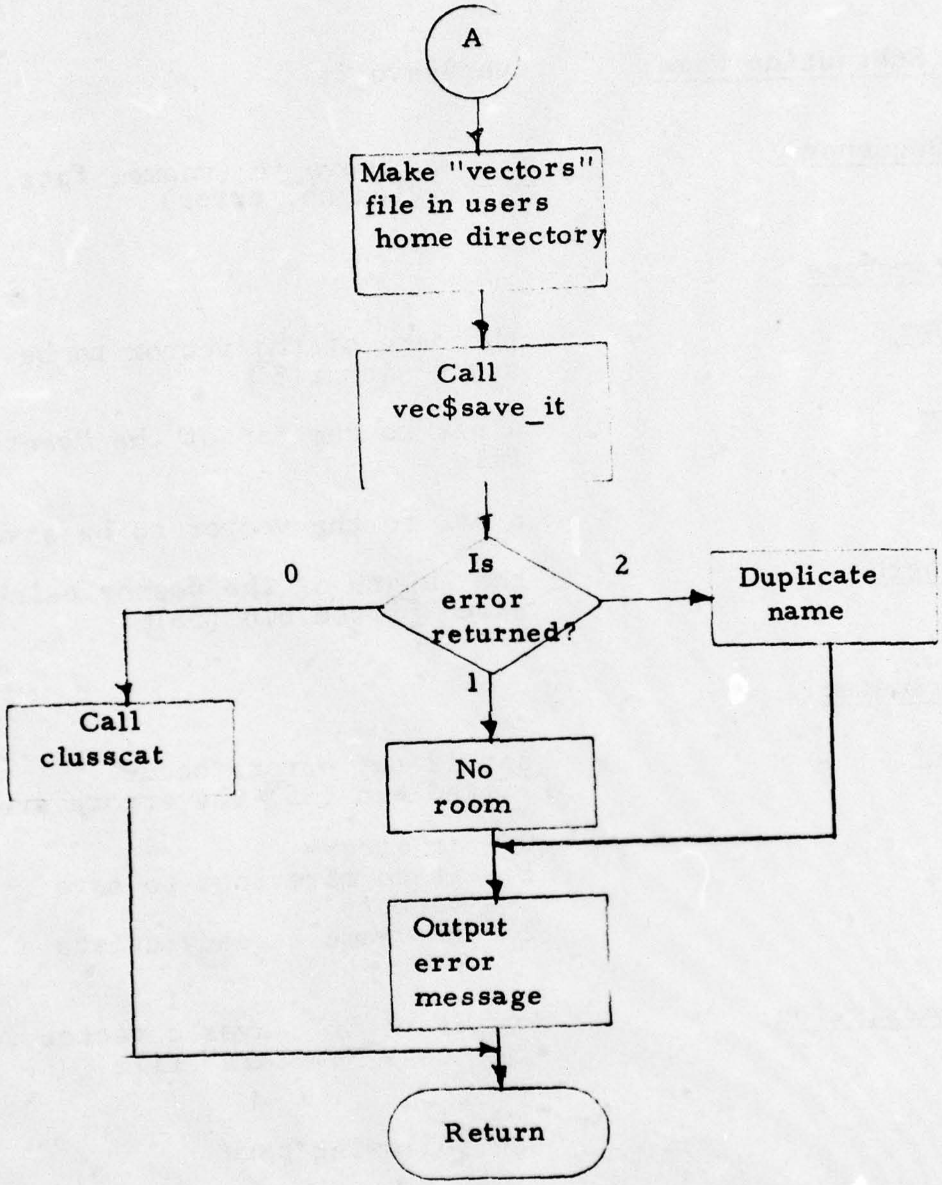
Program Description: vec$save saves the current projection vector in the users vectors file. If a scatter-cluster display is a current display, it asks the user whether he wants to save the x or the y projection vector or both.

Flow Chart: See following page

vec$save



Start
Initiate display file
Does it exist?
no
yes
Is current display cluster_scatter or histogram?
no
yes
Output error message
Return
Is it histo?
yes
no
Get ndim
Get ndim & proj vector index
Does user want to save y proj.
no
Get index to x proj.
yes
Get index to y proj
A


A
Make "vectors" file in users home directory
Call vec$save_it
Is error returned?
0
2
1
Duplicate name
Call clusscat
No room
Output error message
Return

Internal Subroutine Name: vec$save_it

Calling Sequence: call vec$save_it (vname, fptr, vptr, vlength, error)

Input Parameters:

vname — the name of the vector to be saved [char(8)]

fptr — a ptr to the top of the "vectors" file

vptr — a ptr to the vector to be saved

vlength — the length of the vector being saved [fixed bin (35)]

Output Parameter:

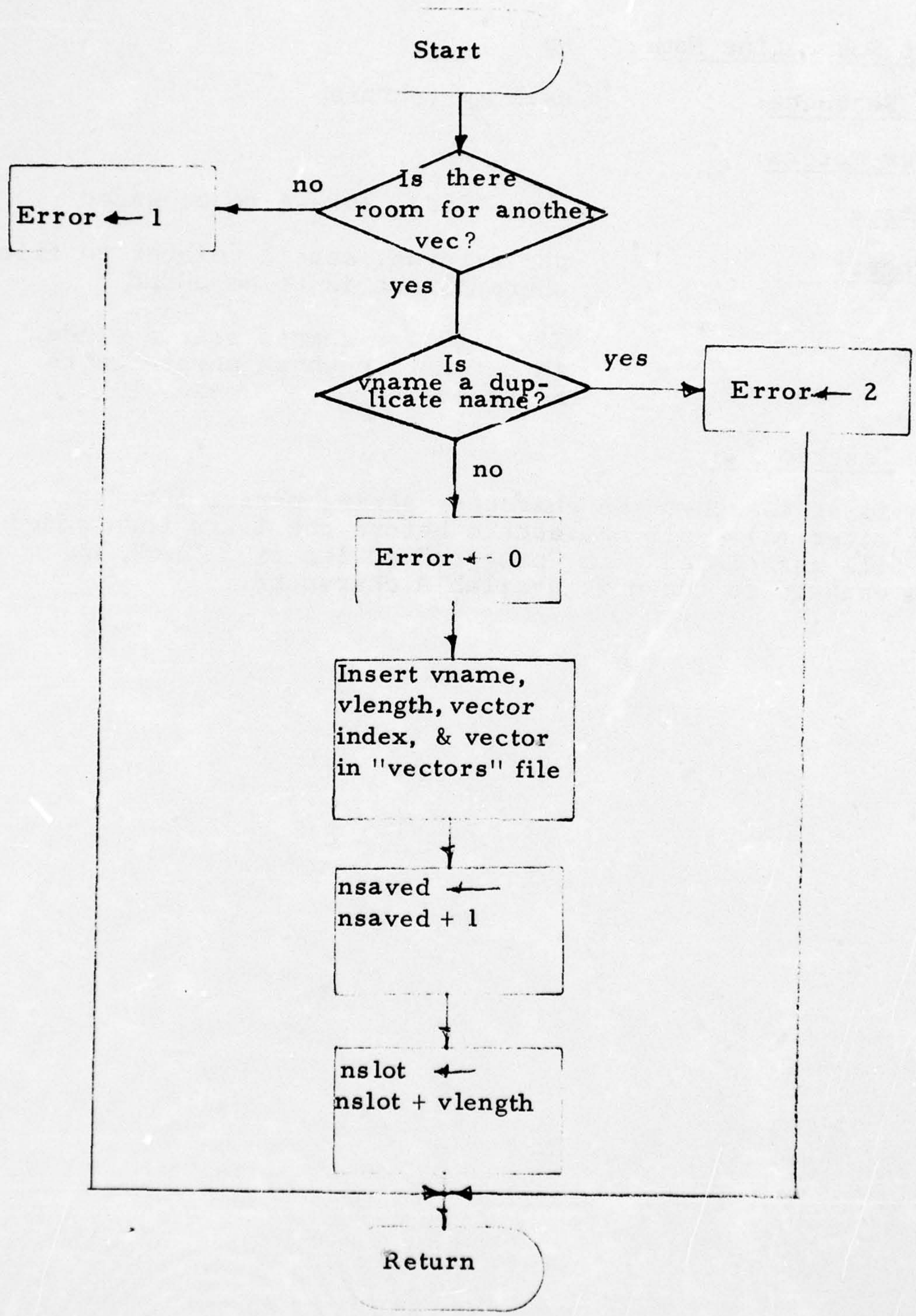
error — set if any errors occur [fixed bin (35)] the errors are:

0 - no errors
1 - if no more room to save vectors
2 - if vname already exists

Program Description: "vec$save_it" saves a vector in the users "vectors" file

Flow Chart: see following page

vec$save_it


Start
Is there room for another vec?
no
Error ← 1
yes
Is vname a duplicate name?
yes
Error ← 2
no
Error ← 0
Insert vname, vlength, vector index, & vector in "vectors" file
nsaved ← nsaved + 1
nslot ← nslot + vlength
Return

Internal Subroutine Name: wp

Calling Sequence: call wp (phrase)

Input Parameters:

phrase	char (168) phrase to be added
progptr	ptr external static pointer to file where phrase is to be added
loc	fixed (35) external static index into the file where phrase is to be added

Program Description:

wp first searches the character string phrase for the end delimiter (%). All characters before the % are then added to the file associated with "progptr" at location "loc", loc is then updated to the next available character.